Analysis of problem **rich_num**

Subtask 1: The most brute force idea is sufficient here - for every number in every query do $O(\sqrt{N})$ checks.
Subtask 2: Last solution is naturally super slow - how do we speed it up? Here we have the added constraint that $V = 10$ always, so a number is either always "rich" or not. So we can precompute the status of every number and seeing the constraint $L = 1$ we can use prefix sums to answer every query in $O(1)$ complexity.
Subtask 3: Here we have some changes in the constraints. First $V \leq 10$, not equals. This is not a big deal, we can simply use 10 prefix arrays. $L = 1$ is also not guaranteed here, but that is not a problem - use can see that $prefix_r - prefix_{l-1}$ still gives the correct answer in $O(1)$ time.

The raise in the limit of R - from 10^4 to 10^5 - should remind us that we still precompute relatively slow - in $O(N\sqrt{N})$ time. The essence of our current idea is that every number "turns behind itself" to check for its divisors. This however creates a lot of useless checks(6 will never divide 49, but we still check it). So let's turn the logic around. Instead of every number searching for its divisors, let every number tell all its multiples that it is their divisor(5 will "tell" 10, 15, 20, 25…). This leaves no space for any "useless" checks and changes the complexity of the precomputation from $O(N\sqrt{N})$ to $O(NlogN)$. Why exactly $O(NlogN)$? This follows from the harmonic series - you can search what that is on the internet.

It is important to notice the weird constraint $V \leq 10$. The exact number is not important, however let's think about what happens when the sum of divisors of a number is greater than 10. No matter the exact V for our query, such numbers will always be "rich".
Subtask 4: Now there are no added constraints. To continue our current solution we must think more about our last observation. We found an opportunity to add a lot of numbers in the group of the "rich" numbers in the same time. What exactly does that mean? Say, we concluded work on a query with some V and the next query has $V - 1$. Then all we have to do is simply add the count of numbers with sum of divisors equal to $V - 1$. But this is would be a really rare occurrence. Well first of all we can chain such operations, so if the next query has $V - x$ no matter x we can deal with that. Also we must notice that the problem is offline - we can sort our queries in a desired order and then rearrange the answers in input order.

How exactly do we "add the count of numbers…"? Well, in the language of our prefix arrays, that would mean changing a bunch of zeroes to ones - "not rich" to "rich" numbers. A hard task for the prefix arrays, but easy for segment trees/Fenwick trees.

Our final complexity will be $O(NlogN)$.

Alternatively, the task can be solved with merge sort tree, but this data structure isn't expected for the competition according to the syllabus.

Author: Ivan Lupov