**Rate Limiting Solution**

This is the solution of the Rate Limiting Problem.

The goal in this problem is to ensure that no more than N requests are made over a window of T seconds. When we are trying to solve a problem over a window of values, where values before the start of the window and after the end of the window become irrelevant, it makes sense to use a sliding window approach. And to implement this approach we will be using a queue data structure, or any similar data structure that supports queue like operations.

Let us add an allowedRequests variable to our class, which will store all the timestamps for when a request was made. The constraints for our allowedRequests are:

* The size (number of allowed requests) cannot exceed N
* The difference between the maximum and minimum timestamps in our queue cannot exceed T

It is fair to say that shouldAllow() will be called each time a request reaches us. So it looks like an optimal place to update allowedRequests if a call is allowed to be made as per our constraints.

Every time we reach a conclusion that the request should be processed, we update allowedRequests with the new timestamp, and we'll add this timestamp at the end of our queue.

We will also ensure that if our allowedRequests is outdated i.e. if it has timestamps for requests older than T seconds, we move our first node in the queue forward, and preferably even delete these timestamps from our allowedRequests queue, and this deletion would also happen in FIFO manner.

Now as you can see, the reason we use a queue data structure here is that the data that came in first (oldest request), will also be the first one to be removed.

At this point, our shouldAllow() function would look like this