Documentation

The problem presents an elimination game where numbers from 1 to n are arranged in increasing order, and a stepwise elimination process is followed. The elimination alternates between left-to-right and right-to-left, removing every second number in each step until only one remains. A naive approach would involve simulating this process by maintaining an actual list, but this quickly becomes inefficient for large values of n. Instead, we can identify patterns in how the first remaining number (head) evolves during each elimination step, leading to an optimized mathematical approach.

To solve the problem efficiently, we track four key variables: head, step, remaining, and left. The head represents the first number remaining in the sequence, the step determines the spacing between numbers in each iteration, the remaining keeps track of the count of numbers left, and the left is a boolean flag indicating the elimination direction. Initially, the head starts at 1, the step at 1, and the remaining at n. With each iteration, if the elimination is from the left or if the number count is odd, the head is updated by adding a step. The remaining count is halved, the step is doubled, and the direction toggles. This ensures an efficient way to track the process without storing the full list.

Analyzing the time complexity, we see that since the number of remaining elements is halved in every step, the total iterations are proportional to O(logn). This logarithmic time complexity makes the approach feasible for large inputs such as $n = 10^9$. Additionally, the space complexity is O(1) since only a few integer variables are maintained rather than an entire list, making the solution optimal in terms of memory usage.

Various edge cases must be considered while implementing this approach. If n = 1, the answer is directly 1 as no elimination occurs. For powers of two, such as n = 16 or n = 32, the elimination follows a predictable sequence. The handling of odd and even values of n ensures that the approach works for all cases. The logarithmic nature of the solution guarantees that it performs efficiently, even for large values, where a brute-force simulation would fail due to excessive time complexity.

A dry run for n = 9 helps to visualize the elimination process. Initially, the list is [1, 2, 3, 4, 5, 6, 7, 8, 9]. The first elimination (left to right) removes every second number, leaving [2, 4, 6, 8]. The second elimination (right to left) reduces it to [2, 6]. Finally, the third elimination (left to right) results in [6], which is the last remaining number. This confirms the correctness of our approach.

Alternative methods could be explored but are less efficient. A brute-force simulation of the elimination process would take O(n) time and O(n) space, making it impractical for large values. Some mathematical approaches based on Josephus-like recurrence relations exist, but they require more complex calculations and are not as straightforward to implement as the iterative pattern-tracking method.

This problem highlights the power of pattern recognition in algorithmic problem-solving. Instead of directly simulating each step, understanding how key variables change allows for a more efficient approach. By focusing on the head position and adjusting it accordingly, we reduce the problem to a series of mathematical updates, making it both time and space-efficient. This demonstrates how computational problems can often be optimized with the right observations and mathematical insights.