Documentation for Counting Bits

Counting the number of 1s in the binary representation of integers from 0 highlights the importance of understanding patterns in binary arithmetic. Binary numbers exhibit recursive relationships, where the count of 1s in the binary form of a number can often be derived from its smaller components. This insight allows us to develop a highly efficient algorithm that avoids recalculating binary representations for every number, making the problem solvable in linear time.

The key intuition behind this problem lies in recognizing how binary numbers evolve. For even numbers, dividing the number by 2 is equivalent to right-shifting the binary digits, which does not change the count of 1s. For odd numbers, the count of 1s is exactly 1 more than the previous even number. This relationship can be leveraged to iteratively compute the results for all numbers from 2 to using dynamic programming, where the results of smaller sub-problems are reused to solve larger ones.

The approach begins by initializing an array ans of size n+1n+1, where each index ii stores the count of 1s in the binary representation of ii. Starting from i=1i = 1, the count for each number is calculated based on whether it is even or odd. If it is even, its count is the same as i//2i // 2, and if it is odd, its count is 1 more than i//2i // 2. This simple rule ensures that each number is processed in constant time, leading to an overall time complexity of O(n).

In terms of complexity, the algorithm is efficient in both time and space. The time complexity is O(n) because each number is processed once with a constant amount of work per iteration. The space complexity is O(n) due to the storage of the lens array, which holds the count of 1s for all numbers up to. This ensures the solution is optimal and scalable, even for large inputs where approaches 10^5.

This approach offers several advantages over naive methods that might involve converting each number to its binary form and counting 1s using string operations. Such methods have a higher time complexity of O(log n) due to the bitwise nature of binary conversions and are less memory efficient. By relying on dynamic programming and binary properties, the proposed solution achieves better performance and simplicity.

Beyond solving this specific problem, the principles used here have broader applications in areas such as cryptography, data compression, and distributed computing, where binary representation plays a crucial role. Understanding how to efficiently manipulate and analyze binary data is a valuable skill in computational problem-solving.

In conclusion, the solution to this problem demonstrates how recognizing patterns and leveraging dynamic programming can transform a computationally expensive task into an efficient one. By focusing on the recursive structure of binary numbers, the algorithm achieves optimal time and space complexity, making it both practical and elegant. This problem serves as an excellent example of how mathematical insights can simplify algorithmic challenges.