Power of Four: Detailed Documentation

The problem revolves around determining whether a given integer n is a power of four. A number is considered a power of four if it can be expressed as 4^x, where x is a non-negative integer. This means n must satisfy specific conditions that make it distinct from other numbers. Unlike general numbers, powers of four exhibit a unique binary structure, which allows for efficient evaluation using bitwise operations.

One of the key observations is that powers of four are a subset of powers of two. Powers of two are characterized by having exactly one bit set in their binary representation (e.g., $2^0 = 1$ as 1, $2^1 = 2$ as 10, $2^2 = 4$ as 100). Powers of four add a constraint: the single set bit must be in an even position when using 1-based indexing. For example, $4^1 = 4$ in binary is 100, where the set bit is in the third position (even index).

To determine if a number n meets these criteria, three conditions must be satisfied. First, n must be positive. Negative numbers and zero cannot be powers of four, as all powers of four are strictly greater than zero. Second, n must be a power of two, verified using the condition n&(n−1)==0. This ensures that n has only one set bit. Finally, the position of this bit must align with powers of four, which can be checked using the bitmask 0x55555555. This mask is designed to cover all numbers with set bits in even positions.

The bitmask 0x555555550 plays a crucial role in optimizing the solution. In its binary form (01010101010101010101010101010101), it ensures that any number ANDed with it will result in a non-zero value only if the set bit is in one of the even positions. This allows the solution to distinguish powers of four from other powers of two without relying on loops or recursion, which makes the approach both efficient and elegant.

From a computational standpoint, the solution is highly efficient. Using bitwise operations ensures a constant time complexity of O(1), as each condition is evaluated independently and in fixed time. The space complexity is also O(1), as no additional data structures or storage are required. These properties make the solution optimal for evaluating large datasets or solving similar problems involving power calculations.

The methodology used here has broader applications beyond this specific problem. Checking powers of numbers using binary properties is a common task in computer science, particularly in areas like memory alignment, data encoding, and optimization. The approach demonstrated here can be extended to other powers, such as powers of eight or sixteen, by modifying the bitmask to target the respective binary positions.

Lastly, the problem underscores the importance of recognizing patterns in number representations, especially in binary. By understanding the structural properties of powers of four, the solution avoids brute-force methods and leverages mathematical insights for efficient computation. This showcases the power of bitwise logic in simplifying complex numerical problems and highlights its utility in real-world applications where performance is critical.