Documentation

1. Problem Statement

The Binary Watch problem involves determining all possible times that can be represented by a binary watch with a given number of LEDs turned on. A binary watch has four LEDs representing the hours (ranging from 0 to 11) and six LEDs representing the minutes (ranging from 0 to 59). The LED's binary state is either on (1) or off (0). The task is to generate all valid times based on the number of LEDs turned on, represented by the integer turnedOn. The hour part of the time must not contain leading zeros (e.g., "01:00" is invalid, "1:00" is valid), while the minute part must always consist of two digits, including a leading zero when necessary (e.g., "10:2" is invalid, "10:02" is valid).

2. Intuition

To solve this problem, we need to consider all possible hour and minute values and check if the number of 1 bits (set bits) in their binary representation matches the given turnedOn value. The goal is to count the number of set bits in both the hour and minute, and if their sum equals turnedOn, we format the time and add it to the results. This requires iterating over all possible hours and minutes, counting the set bits for each combination, and ensuring that the time is correctly formatted.

3. Key Observations

1. **Limited Range for Hours and Minutes:**
   The possible hours range from 0 to 11 (i.e., four bits are needed to represent the hours), and the possible minutes range from 0 to 59 (requiring six bits). This provides a fixed range of values to explore, making the solution bounded and manageable.

2. **Counting Set Bits:**
   The core of the solution is to count the number of 1 bits in the binary representation of both the hours and the minutes. If the total number of set bits across both is equal to turnedOn, the combination is a valid time.

3. **Formatting Requirements:**
   The time needs to be formatted such that the hour is not padded with leading zeros, while the minute is always shown as two digits. This ensures that the output adheres to the expected format.

4. Approach

The solution involves iterating over all possible hours (from 0 to 11) and minutes (from 0 to 59). For each combination of hour and minute, the number of 1 bits in their binary representation is computed using Python's built-in bin() function, which converts a number to its binary representation. The total number of 1 bits from both the hour and minute is then checked against the turnedOn value. If they match, the time is formatted as a string in the format hour:minute and added to the result list. The minute is formatted to ensure it has two digits, even if the value is less than 10.

5. Edge Cases

Several edge cases need to be considered:

- **When turnedOn = 0:**
  This case should only return the time "0:00", as no LEDs are turned on. It tests whether the solution handles the lower bound of the input correctly.
- **When turnedOn = 10:**
  Since there are only 10 LEDs in total (4 for hours and 6 for minutes), it is impossible to have 10 LEDs turned on. This should return an empty list, which verifies that the solution correctly handles the upper bounds of the input.
- **General Cases with Intermediate Values:**
  When turnedOn is within the range of 0 to 9, we need to account for multiple valid times, ensuring that each is correctly formatted and counted.

6. Complexity Analysis

- **Time Complexity:**
  The time complexity is **O(720)**, as we are iterating over all possible hour values (12 values) and all possible minute values (60 values). For each combination, we count the set bits, which is a constant time operation. Therefore, the overall complexity remains manageable even with the largest input sizes.

- **Space Complexity:**
  The space complexity is **O(1)** for the space used by the input and intermediate variables. The space used for storing the result list depends on the number of valid times found, but this is limited by the total possible combinations (720 in the worst case). Thus, the space complexity is considered constant for practical purposes.

7. Alternative Approaches

While the current approach is straightforward and efficient, alternative methods could include backtracking or bitmasking. However, these approaches would add unnecessary complexity without providing significant performance gains. Backtracking might involve exploring combinations of set bits for each LED, while bitmasking could represent each possible hour and minute combination as a bitmask and count the 1 bits. Both approaches are viable but would likely make the solution more complicated without a corresponding improvement in efficiency.

8. Code Implementation

The solution involves iterating over the possible hours and minutes, counting the set bits in their binary representations, and checking if their sum equals turnedOn. If it does, the time is formatted and added to the result. The solution ensures that the hour part has no leading zeros, while the minute part is always formatted as two digits.

9. Test Cases

The solution has been tested with various test cases, including edge cases such as when turnedOn = 0 (expecting ["0:00"]) and when turnedOn = 10 (expecting an empty list). Additional test cases with intermediate values of turnedOn produce valid times according to the binary representation rules.

- **Test Case 1:** turnedOn = 1
  Expected Output: ["0:01", "0:02", "0:04", "0:08", "0:16", "0:32", "1:00", "2:00", "4:00", "8:00"]
- **Test Case 2:** turnedOn = 9
  Expected Output: []
- **Test Case 3:** turnedOn = 0
  Expected Output: ["0:00"]
- **Test Case 4:** turnedOn = 10
  Expected Output: []

10. Final Thoughts

This problem is a good exercise in binary manipulation and bit counting. The solution is efficient and handles all edge cases well, including the minimum and maximum possible values for turnedOn. The approach leverages simple iteration over a fixed range of values and ensures the correct formatting of the time. Although there are alternative approaches, the iterative method provides an elegant and easy-to-understand solution for this problem. The complexity is well within acceptable bounds, making this solution suitable for the given constraints.