Documentation

The missingNumber function in this solution addresses the problem of finding the missing number in an array containing (n) distinct integers ranging from (0) to (n). The array includes all numbers within the specified range except for one missing number, which we aim to identify efficiently. Given constraints on time and space complexity, the solution employs a mathematical approach rather than iterating multiple times or using extra memory, achieving an optimal (O(n)) runtime with (O(1)) space.

The function leverages the well-known formula for calculating the sum of the first (n) natural numbers. This formula, (n * (n + 1) /2), allows us to compute what the total sum would be if no numbers were missing from the array. By using this sum, called expected_sum, we can establish the baseline of all numbers from (0) to (n). To find the missing number, we calculate the sum of the numbers in the array, actual_sum, by summing up all the elements within nums. Since nums include all values except the missing one, the discrepancy between expected_sum and actual_sum will reveal the missing number.

In the function implementation, we first calculate n, which is the array's length and the largest possible number within the range of the array. This value of (n) serves as a boundary for calculating expected_sum. By substituting (n) into the formula, we get the total sum for numbers (0) to (n). Subsequently, we calculate actual_sum by simply summing up all the elements in nums using Python’s built-in sum() function. This step provides the sum of the array elements, capturing all present numbers within the range, except for the one missing value.

The final step in the function is to subtract actual_sum from expected_sum. Since expected_sum includes the missing number, while actual_sum does not, the difference between these two values is precisely the missing number. This subtraction operation completes the function and provides an efficient way to identify the missing integer without additional space. By structuring the solution in this way, we ensure that the function adheres to the optimal constraints given in the problem description.

This approach is efficient both in terms of time and space. The time complexity is (O(n)), which results from summing the elements in the array once, and the space complexity remains (O(1)), as we only use a constant amount of memory for the sum calculations and intermediate variables. Thus, the function meets the problem’s challenge of finding the missing number with minimal computational overhead, making it suitable for large arrays as specified by the constraints.