Documentation

1. Problem Statement:

Given two non-negative integers represented as strings, num1, and num2, return the sum of num1 and num2 as a string. You cannot use any built-in libraries to handle large integers (such as BigInteger) and cannot directly convert the inputs into integers. Your task is to add the strings manually as we do in basic arithmetic.

2. Intuition:

The problem can be approached by simulating the process of adding two numbers digit by digit. The key idea is to start from the least significant digit (rightmost) of both strings, perform the addition, and keep track of any carry that might be generated during the addition. This process is similar to manual addition where you move from right to left, add the digits, and handle the carry accordingly.

3. Key Observations:

- **No direct integer conversion**:
  - We cannot convert the strings directly into integers.
- **Carry Handling**:
  - Whenever the sum of two digits exceeds 9, we need to carry over the excess to the next higher position.
- **Digit-wise processing**:
  - Since we are adding strings, we need to process the strings one digit at a time, from the rightmost digit.
- **No leading zeros**:
  - Both num1 and num2 have no leading zeros, except for the case where the number is zero itself.

4. Approach:

To solve the problem, we can follow these steps:

a. **Reverse Traversal**:

- Initialize two pointers (i and j) to point to the last digits of num1 and num2, respectively. This allows us to traverse the strings from right to left, just like we do when adding numbers manually.

b. **Sum digits**:

- At each step, extract the digits from both strings. If either string is exhausted, treat the corresponding digit as 0. Add the digits along with any carry from the previous edition.

c. **Handle Carry**:

- If the sum of two digits is greater than or equal to 10, compute the carry (the number to be added to the next higher position), and append the current digit to the result list.

d. **Continue until all digits are processed**:

- Repeat the process until all digits from both strings are processed, including any remaining carry.

e. **Reverse and Return**:

- Since we build the result from the least significant digit to the most significant, the result will be in reverse order. Reverse the result list and return it as a string.

5. Edge Cases:

- **Zero Input**: If one of the inputs is "0", the result should simply be the other input.
- **Different Lengths**: If num1 and num2 have different lengths, the shorter string should be treated as having leading zeros.
- **Carry Left**: If there's a carry left after processing all digits, it should be appended to the result.
- **Maximum Length**: The lengths of num1 and num2 can go up to 10^4, so our solution must be efficient enough to handle strings of this length.

6. Complexity Analysis:

a. Time Complexity:

- The solution iterates through both strings once. The while loop runs for the length of the longest string, which is at most 10^4. Hence, the time complexity is **O(max(m, n))**, where m and n are the lengths of num1 and num2.

b. Space Complexity:

- The space complexity is determined by the space required to store the result string. In the worst case, the result can have a length equal to the length of the longer string, plus one extra digit for the carry. Therefore, the space complexity is **O(max(m, n))**.

7. Alternative Approaches:

a. BigInteger (Not allowed):

We could use libraries like BigInteger in Java or arbitrary precision integers in Python to directly convert the strings to integers and perform addition. However, this approach violates the problem constraints as it involves directly handling large integers.

b. Simulating Addition with Stack:

Another approach could involve using stacks to store intermediate results and then building the final answer by popping the values from the stacks, though this approach would involve more complex manipulations and is not as direct as the one provided here.

8. Test Cases:

Test Case 1:

- ***Input:*** num1 = "11", num2 = "123"
- ***Output:*** "134"
- ***Explanation:*** Adding 11 and 123 gives 134.

Test Case 2:

- ***Input:*** num1 = "456", num2 = "77"
- ***Output:*** "533"
- ***Explanation:*** Adding 456 and 77 gives 533.

Test Case 3:

- ***Input:*** num1 = "0", num2 = "0"
- ***Output:*** "0"
- ***Explanation:*** Adding two zeros results in zero.

Test Case 4:

- ***Input:*** num1 = "999", num2 = "1"
- ***Output:*** "1000"
- ***Explanation:*** Adding 999 and 1 gives 1000.

Test Case 5:

- ***Input:*** num1 = "9999", num2 = "9999"
- ***Output:*** "19998"
- ***Explanation:*** Adding 9999 and 9999 gives 19998.

9. Final Thoughts:

The solution effectively handles the addition of large numbers represented as strings by simulating the manual addition process. The time and space complexity are both linear in terms of the length of the input strings, making it efficient enough for the input size constraint of up to 10^4. The edge cases are well-managed, ensuring that the algorithm works even for zeros, differing lengths, and carry-over situations.