Complete Documentation for the twoSum Function

Problem Description

Given a 1-indexed array of integers numbers that is sorted in non-decreasing order, find two numbers in the array such that they add up to a specific target number. *The function should return the indices of the two numbers (index1 and index2) as an array of length 2, where:*

1 <= index1 < index2 <= numbers.length

The solution must ensure that it uses only constant extra space and runs in linear time (O(n)).

Function Signature

```
def twoSum(self, numbers: List[int], target: int) -> List[int]:
```

Parameters

- **numbers (List[int]):** A list of integers sorted in non-decreasing order. The list is 1-indexed, meaning that the first element is considered at index 1 for the purpose of this problem.

Constraints:

- 2 <= numbers.length <= 30,000
- -1000 <= numbers[i] <= 1000
- The input list is guaranteed to have exactly one solution.

- **target (int):** An integer representing the target sum that we need to find by adding two distinct numbers from the input list numbers.

Constraints:

- -1000 <= target <= 1000

Returns

- **List[int]:** *A list containing two integers [index1, index2]:*
  - index1 and index2 are the 1-based indices of the two numbers in numbers that add up to target.
  - The returned indices should satisfy 1 <= index1 < index2 <= numbers.length.

Methodology

To solve this problem using the two-pointer technique, follow these steps:

1. **Initialize Two Pointers:**
   - left: Start at the beginning of the list (0 in 0-based indexing).
   - right: Start at the end of the list (len(numbers) 1 in 0-based indexing).

2. **Iterate Over the List:**
   - Calculate the sum of the elements at the current positions of left and right.
   - If the sum equals the target, return the indices left + 1 and right + 1 (converted to 1-based indexing).
   - If the sum is less than the target, increment the left pointer to increase the sum.
   - If the sum is greater than the target, decrement the right pointer to decrease the sum.

3. **End Condition:**
    - The loop continues until left is no longer less than right. The problem guarantees that a solution always exists, so the function will always find and return a valid pair of indices.

Example 1:

- **Input:** numbers = [2, 7, 11, 15], target = 9
- **Output:** [1, 2]
- **Explanation:** The sum of numbers[1] and numbers[2] (1-based index) is 2 + 7 = 9. Hence, the function returns [1, 2].

Example 2:

- **Input:** numbers = [2, 3, 4], target = 6
- **Output:** [1, 3]
- **Explanation:** The sum of numbers[1] and numbers[3] (1-based index) is 2 + 4 = 6. Hence, the function returns [1, 3].

Example 3:

- **Input:** numbers = [-1, 0], target = -1
- **Output:** [1, 2]
- **Explanation:** The sum of numbers[1] and numbers[2] (1-based index) is -1 + 0 = -1. Hence, the function returns [1, 2].

Edge Cases

1. **Minimum Length Input:**
   - The input list has a length of 2, which is the minimum allowed by the constraints. The solution will directly return [1, 2] since the only two elements must sum to the target.

2. **Negative Numbers:**
   - The list can contain negative numbers. The function correctly handles both positive and negative integers, as well as zeros.

3. **Target Equals One of the Elements:**
   - The function will correctly find the pair that sums to the target even if the target equals one of the elements (e.g., numbers = [1, 2, 3, 4], target = 3).

Complexity Analysis

- **Time Complexity:** O(n), where n is the length of the input list numbers. This is because each element is processed at most once due to the two-pointer approach.

- **Space Complexity:** O(1). The solution uses only a constant amount of extra space, as required by the problem constraints.

This solution is both time-efficient and space-efficient, making it well-suited for large input sizes.