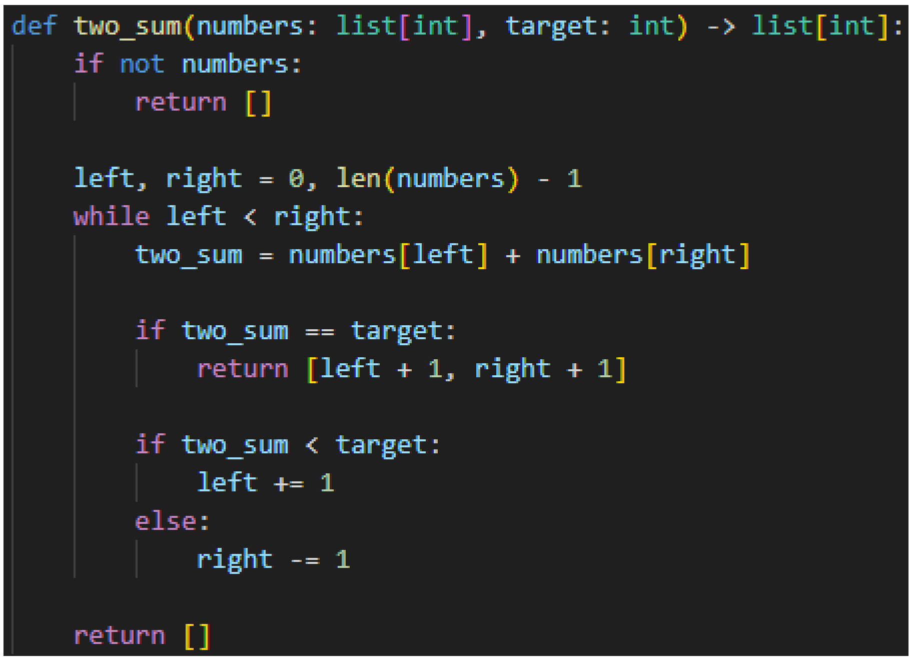
Problem Statement:

Given a sorted list of integers (numbers) in non-decreasing order and a target integer (target), find the indices (1-indexed) of two numbers in the list that add up to the target. The solution must use O(1) additional space and ensure that the first index is less than the second.

Solution 1



```python
def two_sum(numbers: list[int], target: int) -> list[int]:
    if not numbers:
        return []

    left, right = 0, len(numbers) - 1
    while left < right:
        two_sum = numbers[left] + numbers[right]

        if two_sum == target:
            return [left + 1, right + 1]

        if two_sum < target:
            left += 1
        else:
            right -= 1

    return []
```

Step-by-Step Breakdown

1. **Input:**
    - numbers: A sorted list of integers, e.g., [2, 7, 11, 15].
    - target: An integer representing the desired sum, e.g., 9.
2. **Intermittent step 1:**
    - Edge Case Check:
        - If numbers is empty, return an empty list [].
3. **Intermittent step 2:**
    - Initialize Two Pointers:
        - Set left to the start index (0).
        - Set right to the last index (len(numbers) - 1).
    - Iterate with Two-Pointer Technique:
        - While left < right, compute the sum of numbers[left] + numbers[right].

        - Compare Sum to Target:
            - If the computed sum equals the target, return [left + 1, right + 1] (converting to 1-indexed positions).
            - If the sum is less than target, increment left to try a larger value.
            - If the sum is greater than target, decrement right to try a smaller value.

4. **Output:**
    - If a valid pair is found, the function returns their 1-indexed positions as a list, e.g., [1, 2].
    - If no such pair exists, the function returns an empty list [].
5. **Efficiency:**
    - Time Complexity: O(n) – Each element is examined at most once.
    - Space Complexity: O(1) – Only a few pointers are used regardless of the input size, meeting the O(1) additional space requirement.

Visual Flow Diagram

```
             Input: numbers = [2, 7, 11, 15], target = 9
                             │
                             ▼
            ┌───────────────────────────┐
            │ Check if numbers is empty │──Yes──> Return []
            └───────────────────────────┘
                             │
                             ▼
         Initialize pointers: left = 0, right = 3
                             │
                             ▼
               ┌─────────────────────────┐
               │ while left < right:     │
               └─────────────────────────┘
                             │
                             ▼
            Compute current_sum = numbers[left] + numbers[right]
                             │
                             ▼
       ┌─────────────────────────────────────────┐
       │ Is current_sum equal to target?         │
       └─────────────────────────────────────────┘
                │                   │
               Yes                  No
                │                   │
                ▼                   ▼
Return [left+1, right+1]   ┌───────────────────────────┐
                           │ Is current_sum < target?  │
                           └───────────────────────────┘
                                    │           │
                                  Yes          No
                                    │           │
                                    ▼           ▼
                              Increment left  Decrement right
                                    │           │
                                    └───Loop────┘
                             (Repeat until left >= right)
                             │
                             ▼
                   No valid pair found → Return []

```