### Algorithm: Two Pointers Technique

This algorithm uses two pointers to solve problems that involve finding pairs or subarrays that satisfy certain conditions. It often results in a time complexity of (O(n)), where (n) is the length of the array.

#### Initialize Data Structures:

* Use two pointers (left and right) to represent the current window or pair.
* Track the conditions or results as needed.

#### Expand and Contract the Window:

* Move the pointers based on the conditions.
* If a condition is met, update the necessary result.

#### Repeat:

* Repeat the process until the pointers traverse the entire array.

#### Implementation:

| **def** two\_sum\_sorted(nums: List[int], target: int) -> List[int]:  left, right = 0, len(nums) - 1  **while** left < right:  current\_sum = nums[left] + nums[right]  **if** current\_sum == target:  **return** [left, right]  **elif** current\_sum < target:  left += 1  **else**:  right -= 1  **return** []  *# Example usage:*  nums = [2, 7, 11, 15]  target = 9  print(two\_sum\_sorted(nums, target)) *# Output: [0, 1]* |
| --- |

#### 

#### Explanation:

Initialize:

* left: The left pointer, initially set to 0.
* right: The right pointer, initially set to the last index of the array.

Iterate with Pointers:

* For each step, calculate the sum of the elements at the pointers.
* If the sum equals the target, return the indices.
* If the sum is less than the target, move the left pointer to the right.
* If the sum is greater than the target, move the right pointer to the left.