The **Two Pointer Technique** is a commonly used algorithmic strategy to **solve problems efficiently on sorted arrays or linked lists**. It involves using two indices (or pointers) that move toward each other (or in the same direction) to find a solution.

**✅ When to Use Two Pointers?**

* The array is **sorted**
* You need to find a **pair of elements** that satisfy a condition (e.g., their sum equals a target)
* You want **O(n)** time instead of O(n²)

**📘 Example Problem:**

**"Given a sorted array numbers[], find two numbers such that they add up to a target value."**

**🔍 Concept Explanation:**

Let’s say:

numbers = [2, 7, 11, 15]

target = 9

We're trying to find two numbers whose **sum is 9**.

### ❗ Brute Force (O(n²)):

Check every pair:

* 2+7=9 ✅
* Done.

This is **slow** for large arrays

### Two Pointer Technique (O(n)):

1. Set:
   * left pointer at start (index 0)
   * right pointer at end (index n-1)
2. Compute sum of numbers[left] + numbers[right]:
   * If sum == target → return
   * If sum < target → increase left (to increase sum)
   * If sum > target → decrease right (to decrease sum)

## ****Code with Line-by-Line Explanation:****

class Solution {

public:

vector<int> twoSum(vector<int>& numbers, int target) {

int left = 0; // left pointer starts at the beginning

int right = numbers.size() - 1; // right pointer starts at the end

while (left < right) { // loop until pointers cross

int sum = numbers[left] + numbers[right]; // compute current sum

if (sum == target) {

// found the pair, return 1-based indices

return {left + 1, right + 1};

} else if (sum < target) {

// sum is too small, move left pointer to the right

left++;

} else {

// sum is too big, move right pointer to the left

right--;

}

}

return {-1, -1}; // if no pair found

}

#### };

#### Logic line by line Step 1:

* left = 0 → numbers[0] = 2
* right = 3 → numbers[3] = 15
* sum = 2 + 15 = 17 → too big → move right-- → now right = 2

#### Step 2:

* left = 0 → numbers[0] = 2
* right = 2 → numbers[2] = 11
* sum = 2 + 11 = 13 → still big → move right-- → right = 1

#### Step 3:

* left = 0 → numbers[0] = 2
* right = 1 → numbers[1] = 7
* sum = 2 + 7 = 9 ✅ found

Return {1, 2} (1-based index)