## LeetCode 167. Two Sum II – Input Array Is Sorted

### 1. Problem Title & Link

- **167. Two Sum II – Input Array Is Sorted**
- https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/

### 2. Problem Statement (Short Summary)

You are given a **1-indexed** array of integers numbers sorted in **non-decreasing order**, and an integer target.
Return the **indices (1-based)** of the two numbers that add up to target.
You may assume exactly **one solution** exists.

### 3. Examples (Input → Output)

Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: 2 + 7 = 9

Input: numbers = [2,3,4], target = 6
Output: [1,3]

Input: numbers = [-1,0], target = -1
Output: [1,2]

### 4. Constraints

- 2 <= numbers.length <= 3 * 10^4
- -1000 <= numbers[i] <= 1000
- numbers is sorted in **non-decreasing order**
- Exactly one valid answer exists

### 5. Thought Process (Step by Step)

💡 Because the array is sorted, we can use **two pointers** to find the pair in O(n) time.

### Approach 1: Two Pointer Technique ❤️

1. Initialize:
   - left = 0
   - right = len(numbers) - 1
2. Compute sum = numbers[left] + numbers[right]
   - If sum == target → return [left+1, right+1]
   - If sum < target → move left++ (need bigger sum)
   - If sum > target → move right-- (need smaller sum)
3. Continue until you find the pair.

### Approach 2: Binary Search (Optional for teaching contrast)

For each element, binary search for the complement (target - numbers[i]).
Time: O(n log n)
But the two-pointer version is **cleaner and faster**.

## 6. Pseudocode

```
left = 0
right = n - 1

while left < right:
   sum = numbers[left] + numbers[right]
   if sum == target:
      return [left + 1, right + 1]
   else if sum < target:
      left += 1
   else:
      right -= 1
```

## 7. Code Implementation

### ✅ Python

```
class Solution:
   def twoSum(self, numbers: List[int], target: int) -> List[int]:
      left, right = 0, len(numbers) - 1

      while left < right:
         s = numbers[left] + numbers[right]
         if s == target:
            return [left + 1, right + 1]
         elif s < target:
            left += 1
         else:
            right -= 1
```

### ✅ Java

```
class Solution {
   public int[] twoSum(int[] numbers, int target) {
      int left = 0, right = numbers.length - 1;

      while (left < right) {
         int sum = numbers[left] + numbers[right];
         if (sum == target)
            return new int[]{left + 1, right + 1};
         else if (sum < target)
            left++;
         else
            right--;
      }
      return new int[]{-1, -1}; // fallback
   }
}
```

## 8. Time & Space Complexity

- **Time:** O(n) — one traversal
- **Space:** O(1) — in-place two-pointer technique

## 9. Dry Run (Step-by-Step Execution)

👉 Input: numbers = [2,7,11,15], target = 9

| Step | left | right | sum | Action | Comment |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 17 | sum > 9 → move right | too large |
| 2 | 0 | 2 | 13 | sum > 9 → move right | still too large |
| 3 | 0 | 1 | 9 | ✅ match found | return [1,2] |

✅ Output: [1,2]

## 10. Concept Insight Table

| Core Concept | Common Use Cases | Common Traps | Builds / Next Steps |
|---|---|---|---|
| **Two-Pointer Technique** — simultaneously scan from both ends to meet condition. | - Sorted array problems - Pair sum / difference finding - Window shrinking logic | - Forgetting array is sorted - Returning 0-based indices (LeetCode expects 1-based) - Overlapping pointers or infinite loops | 🔷 Builds to **LeetCode 15 (3Sum)**, **18 (4Sum)** 🔷 Connects to **sliding window** problems 🔷 Strengthens “left-right compression” thinking |

## 11. Common Mistakes / Edge Cases

- Returning [left, right] instead of [left+1, right+1].
- Using hash map — unnecessary (works but not optimal for sorted input).
- Forgetting to move correct pointer based on comparison.

## 12. Variations / Follow-Ups

- **Two Sum (LC 1)** → unsorted version (use hash map).
- **3Sum (LC 15)** → add another pointer layer.
- **Two Sum – Input BST (LC 653)** → apply same logic in tree traversal.