0003\_Easy\_TwoSumII\_#167\_Breakdown

Problem:

Given a **1-indexed** array of integers numbers that is already **sorted in non-decreasing order**, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.

Return the indices of the two numbers, index1 and index2, ***added by one*** as an integer array [index1, index2] of length 2.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Examples:

**Example 1:**

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

**Output:** [1,2]

**Explanation:** The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

**Example 2:**

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

**Output:** [1,3]

**Explanation:** The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].

**Example 3:**

**Input:** numbers = [-1,0], target = -1

**Output:** [1,2]

**Explanation:** The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Observations:

* The indexes returned are plus one.
* Must be in place.
* There is exactly one solution.
* The same element cannot be used twice.

What needs to be true for this problem to work:

* The sum of two elements within the array are equal to a given target value.

How would a person solve it:

* Look at the first element of the array, keep it in mind and look through the rest of the array for a number that adds to it to sum to target. If no number is found, move to the second element of the array and repeat.

Brute force:

* For each element in the array, loop through the array checking each other element for a number that adds to target.
  + Time Complexity: O(N^2), for each element of the array, the array is looped through.
    - Specifically O(N(N-1)/2), for each element of the array, the loop through the array decreases in elements because more and more of the beginning of the array had been checked.
  + Space Complexity: O(1), No extra space is needed.

Optimize (BUD, bottleneck, unnecessary code, duplicate code:

* The array is sorted, so instead of doing a full iteration for each element, a more efficient iteration can be done with two pointers instead.
* Start at the beginning of the array, set a point at the end of the array, if the sum of the two numbers is greater than the target, move the end point to the left (which decreases the sum). If the sum is less than the target, move the starting point to the right (increasing the sum). Repeat until a target is found.
* Time Complexity: O(N), the array is iterated through at most once, because the pointers are moving towards each other.
* Space Complexity: O(1), the only extra space needed is three integers (low, high, and sum), which is negligible.

Pseudocode:

* Set an integer called low equal to the first index of the array and an integer called high equal to the last index of the array.
* While low is less than high.
  + Create an integer equal to the sum of low plus high.
  + If sum equals target
    - Return low plus one and high plus one
  + Else if sum is less than target
    - Increase low by one.
  + Else sum is greater than target
    - Decrease high by one
* After the loop is complete, if no solution is found, return null.