# Two Sum 2 - Input Array is Sorted

## 167. Two Sum II - Input Array Is Sorted

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.

**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, i
ndex2 = 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].
```

**Constraints:**

- `2 <= numbers.length <= 3 * 104`
- `1000 <= numbers[i] <= 1000`
- `numbers` is sorted in **non-decreasing order**.
- `1000 <= target <= 1000`
- The tests are generated such that there is **exactly one solution**.

Initial Thought

- Use a variable to store complementary numbers
- We use two pointers at both ends of the array that moves along the array under certain conditions
- If perhaps the index is more than the target, then 2nd pointer shifts to the left, where as index less than the target than the 1st pointer is too small and therefore shifts to the right

Take the first example

- When 2 + 15 = 17, its too big, so when do we decide left or right pointer shifts?

In this case right pointer is too large as no numbers can fit with 15 to make 9 (unless you're going into the negatives, but we assume the array to be sorted)

- This logic would apply to numbers smaller than the target, because the left pointer is too small.

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[in
        x1 = 0
        x2 = len(numbers) - 1

        while x1 < x2:

            current_sum = numbers[x1] + numbers[x2]

            if current_sum == target:
                return [x1 + 1 , x2 + 1]

            elif current_sum > target:
                x2 -= 1

            else:

                x1 += 1

        return[]
```