Two Sum

167. Two Sum II - Input Array Is Sorted Solved

Medium | Topics | Companies

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, index2 = 3. We return [1, 3].
```

Example 3:

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

return any 2 indices, which results the sum of both indices equals to the given target

2 pointer's technique

If you are required to find elements that are satisfying certain constraints or conditions → you use 2 pointers technique

Most Commonly used problems :

1. Two Sum
2. Three Sum , four sum, closest three sum ....
3. Container with most water
4. Sort n Searching
5. String manipulations
6. Palindrome problems
7. LinkedList problems
8. Sliding window problems

So, as our array is sorted , we can make observations out of it

arr : (2, 7, 11, 15)

If you set pointer → i at 0 , j at n-1 ⇒ (2, 7, 11, 15) with i at 2 and j at 15

* if you sum up arr[i] & arr[j] → you will get some 'x' ans right ?

let us assume , (Sum we got == X)

Case - I

If (sum < target) ↴

it means our sum need to be increased to match target value

for that, you have to increment i, as your array is sorted, if you move i forward, the value definetly increase

Case - II

If (sum > target) ↴

it means our sum need to be decreased to match target value

for that, you have to decrement j, as your array is sorted, if you move j backward, the value definetly decreases

Case - III

if (sum == target)

↳ return the index values of (i, j)

	i	j	sum	target	operation
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	0	3	17	< 18	i++
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	1	3	21	> 18	j--
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	1	2	18	== 18	✓ (return {1, 2})