**167. Two Sum II**

<https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/>

1. **Listen**

**Problem Statement:**

Given a **one-dimensional** **array** of **integers** that are 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.*

**Input**:

**one-dimensional array** of integers *numbers*. *numbers* is **sorted in non-decreasing order**

a target number

**Goal**:

find two numbers in the array *numbers* such that they add up to a specific target number

**Return**:

the indices of two numbers in the array numbers that add up to the target number added by one as an integer array.

1. **Example**

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].

**Constraints:**

* There is **exactly one solution**.
* You **may not** use the same element twice.
* *numbers* is sorted in non-decreasing order
* There are two numbers in the input array *numbers*.

Let these numbers be numbers[index1] and numbers[index2]

where

1 <= index1 < index2<= numbers.length

* Your solution must use only constant extra space.
* 2 <= numbers.length <= 3 \* 104 (numbers must be at least 2 length)
* -1000 <= numbers[i] <= 1000
* -1000 <= target <= 1000

**Test Cases:**

* Negative numbers
* Positive numbers
* Mixed positive and negative numbers
* Only two numbers in the array
* More than two numbers in the array

**Edge Cases:**

* Empty array

1. **Brute Force**

**Solution 1: Binary Search Time = O(nlogn) Space = O(logn)**

The fact that the array is sorted in non-decreasing order means we can use binary search.

Use binary search to see if we can find a value that is equal to (target - our current index)

This solution takes O(nlogn) time and O(1) space (O(logn) space if done recursively)

1. **Optimize**

**Solution 2: Two pointer**

The fact that the array is sorted in non-decreasing order also means we can use our two-pointer technique, where we have pointers starting at opposite ends of the array.

We move the front/back pointer depending on if the front and back pointer values add up to the target.

This solution takes O(n) time and O(1) space.

1. **Walkthrough**

while(front < back)

if front + back is equal to target, then return the indices of front+1 and back+1 as a new array of length 2

if front + back is greater than to target, move front pointer back one index

(this makes the possible sum of front and back smaller)

if front + back is less than target, move front pointer back one index

(this makes the possible sum of front and back larger)

1. **Implement**

public int[] twoSum(int[] numbers, int target)

{

int front = 0, back = numbers.length – 1;

while(front < back)

{

if(numbers[front] + numbers[back] == target)

return new int[] {front+1, back+1};

else if(numbers[front] + numbers[back] < target)

front++;

else

back--;

}

return null;

}

1. **Test**

* Negative numbers

Works, because were simply adding numbers together, it doesn’t matter if they’re negative.

* Positive numbers

Works, because were simply adding numbers together, it doesn’t matter if they’re positive.

* Only two numbers in the array

If there’s only two numbers in the array, the function should be done on its first iteration.

* More than two numbers in the array

[2,7,11,15], target = 9

Works when walking through the example

* Empty array

Doesn’t fit the problem constraints, but still works