[167. Two Sum II - Input Array Is Sorted](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/)

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]

**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**.

Solution:

Approaches

1. Brute force

Brute force approach uses two for loops where i variable for loop is on left side of current element and j for loop variable loop through to get remaining sum of the target

This approach takes O(N2) time with Constand extra space to solve

class Solution {

public int[] twoSum(int[] nums, int target) {

int n = nums.length;

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (nums[i] + nums[j] == target) {

return new int[]{i, j};

}

}

}

return new int[]{}; // No solution found

}

}

1. Using Hashmap

Intuition behind hashmap is that to find currents other half in the hashmap or else to store that value to use by other half when it Is needed

class Solution {

public int[] twoSum(int[] nums, int target) {

Map<Integer, Integer> numMap = new HashMap<>();

int n = nums.length;

// Build the hash table

for (int i = 0; i < n; i++) {

numMap.put(nums[i], i);

}

// Find the complement

for (int i = 0; i < n; i++) {

int complement = target - nums[i];

if (numMap.containsKey(complement) && numMap.get(complement) != i) {

return new int[]{i, numMap.get(complement)};

}

}

return new int[]{}; // No solution found

}

}

Best approach

TwoPointers

Intuition behind this approach is that we use two pointer called i and j where I starts from o index and j from end index when i+j == to target we return if i+j is greater than target we reduce the index of j as array is sorted and is is applicable vice versa for I as array is sorted

class Solution {

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

        int i=0;

        int j=numbers.length-1;

        while(i<j){

            if(numbers[i]+numbers[j]==target){

                return new int[]{i+1 , j+1};

            }else if(numbers[i]+numbers[j] > target){

                j--;

            }else{

                i++;

            }

        }

    return new int[]{-1,-1};

    }

}