167. Two Sum II - Input Array is Sorted ↗ (/problems /two-sum-ii-input-array-is-sorted/)

March 12, 2016 | 26.6K views

Average Rating: 4.67 (30 votes)

Given an array of integers that is already **sorted in ascending order**, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

Note:

- Your returned answers (both index1 and index2) are not zero-based.
- You may assume that each input would have *exactly* one solution and you may not use the *same* element twice.

Example:

```
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.
```

Solution

Approach 1: Two Pointers

Algorithm

We can apply Two Sum's solutions (https://leetcode.com/articles/two-sum/) directly to get $O(n^2)$

time, $O(1)$ space using brute force and $O(n)$ time, $O(n)$ space using hash table. However, both existing solutions do not make use of the property where the input array is sorted. We can do better.

We use two indexes, initially pointing to the first and last element respectively. Compare the sum of these two elements with target. If the sum is equal to target, we found the exactly only solution. If it is less than target, we increase the smaller index by one. If it is greater than target, we decrease the larger index by one. Move the indexes and repeat the comparison until the solution is found.

Let $[..., a, b, c, ..., d, e, f, ...]$ be the input array that is sorted in ascending order and the element b, e be the exactly only solution. Because we are moving the smaller index from left to right, and the larger index from right to left, at some point one of the indexes must reach either one of b or e. Without loss of generality, suppose the smaller index reaches b first. At this time, the sum of these two elements must be greater than target. Based on our algorithm, we will keep moving the larger index to its left until we reach the solution.
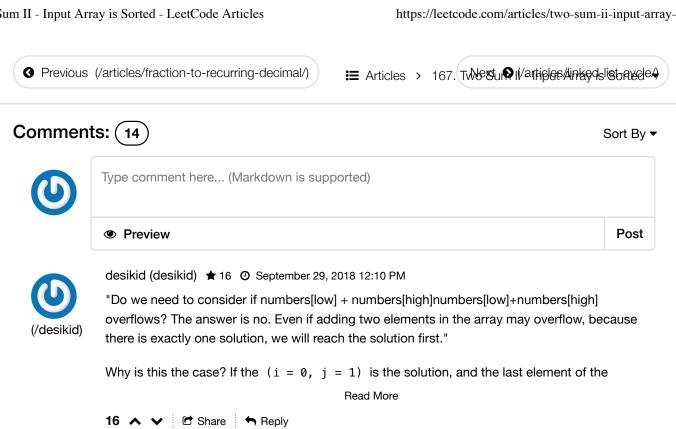
```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        int low = 0, high = numbers.size() - 1;
        while (low < high) {
            int sum = numbers[low] + numbers[high];
            if (sum == target)
                return {low + 1, high + 1};
            else if (sum < target)
                ++low;
            else
                --high;
        }
        return {-1, -1};
    }
};
```

Do we need to consider if $numbers[low] + numbers[high]$ overflows? The answer is no. Even if adding two elements in the array may overflow, because there is exactly one solution, we will reach the solution first.

Complexity analysis

- Time complexity : $O(n)$. Each of the n elements is visited at most once, thus the time complexity is $O(n)$.

- Space complexity : $O(1)$. We only use two indexes, the space complexity is $O(1)$.

Rate this article:

Previous (/articles/fraction-to-recurring-decimal/)

☰ Articles > 167. Two Sum II - Input Array is Sorted ▾

Next (/articles/linked-list-cycle/)

Comments: (14)

Sort By ▾

Type comment here... (Markdown is supported)

👁 **Preview**

Post

(/desikid)

desikid (desikid) ★ 16 🕐 September 29, 2018 12:10 PM

"Do we need to consider if numbers[low] + numbers[high]numbers[low]+numbers[high] overflows? The answer is no. Even if adding two elements in the array may overflow, because there is exactly one solution, we will reach the solution first."

Why is this the case? If the (`i = 0, j = 1`) is the solution, and the last element of the

Read More

16 ⋀ ⋁ ⋮ ⤴ Share ⋮ ↩ Reply

(/dylondickinson)

dylondickinson (dylondickinson) ★ 10 🕐 September 1, 2018 9:22 PM

I don't understand why overflow is not an issue.

7 ⋀ ⋁ ⋮ ⤴ Share ⋮ ↩ Reply

SHOW 2 REPLIES

(/keremsahin)

keremsahin (keremsahin) ★ 24 🕐 February 16, 2018 1:42 AM

You have to consider overflow case. Doesn't work for a test case of numbers={1, 2, 3, 2147483647}, target=3.

7 ⋀ ⋁ ⋮ ⤴ Share ⋮ ↩ Reply

(/xi11)

xi11 (xi11) ★ 186 🕐 October 6, 2017 4:10 PM

I believe binary search is still applicable to this question. I read some existing posts. Most of them used binary search for one value, and their algorithm could be described as following:

```
for(i = 0; i < nums; i++)
    num1 = nums[i];
```

Read More

5 ⋀ ⋁ ⋮ ⤴ Share ⋮ ↩ Reply

SHOW 1 REPLY

sushanth (sushanth) ★ 3 ⏱ June 22, 2018 1:27 PM

☰ Articles › 167. Two Sum II - Input Array is Sorted ▼

class Solution {

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

HashMap<Integer,Integer> hmap = new HashMap<Integer,Integer>();

int diff = 0;

int[] arr = new int[2];

Read More

3 ∧ ∨   ⤴ Share   ↩ Reply

daxios (daxios) ★ 7 ⏱ September 21, 2018 6:50 AM

Intuitively i feel the algorithm is correct, but i can't prove it.

For eg, when nums[low] + nums[high] < target, you can also increase the larger index by 1. So why do you choose to increase the smaller index by 1?

Can we prove that this algorithm won't miss any solution ?

3 ∧ ∨   ⤴ Share   ↩ Reply

SHOW 1 REPLY

meganlee (meganlee) ★ 693 ⏱ September 12, 2018 9:01 PM

Translated into Java version

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int lo = 0, hi = numbers.length - 1;
```

Read More

1 ∧ ∨   ⤴ Share   ↩ Reply

RobotBandit (robotbandit) ★ 5 ⏱ October 10, 2018 8:25 PM

There is a better solution with run time `O(log N)` for distinct sorted integer arrays and guaranteed solution, assuming target << N. Idea is to first lookup for target / 2 with binary search, and then expand window in both directions up to target / 2. With target being << N, this algorithm is `O(log N)`

Read More

0 ∧ ∨   ⤴ Share   ↩ Reply

SHOW 2 REPLIES

RayHuang1993 (rayhuang1993) ★ 0 ⏱ September 8, 2018 8:41 AM

For the overflow issue, may we use a long to save the sum?

0 ∧ ∨   ⤴ Share   ↩ Reply

japoorva88 (japoorva88) ★ 5 ⏱ April 5, 2018 10:41 AM

Javascript:

(/japoorva88)

```
var twoSum = function(numbers, target) {
    for (let i=0 ; i< numbers.length ; i++) {
        let diff = target – numbers[i]
```

Read More

0 ⌄ ⌄   ⌇ ↪ Share ⌇ ↩ Reply

SHOW 1 REPLY

‹ ① ② ›

Help Center (/support/) | Students (/students) | Terms (/terms/) | Privacy