167. Two Sum II - Input Array is Sorted

March 12, 2016 | 65.7K views

Average Rating: 4.59 (46 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 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.

C++ Copy

```
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 Next

Comments: 23

Sort By

Type comment here... (Markdown is supported)

Preview Post

desikid ★ 35 September 30, 2018 12:40 AM

"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 `{1 = 0, 1 = 1}` is the solution, and the last element of the array is a very.

Read More

35 Share Reply

SHOW 1 REPLY

xi11 ★ 239 October 7, 2017 4:40 AM

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++)
```

Read More

24 Share Reply

SHOW 4 REPLIES

leetcoder37 ★ 46 February 16, 2018 3:12 PM Report

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

13 Share Reply

SHOW 2 REPLIES

dylondickinson ★ 15 September 2, 2018 9:52 AM

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

9 Share Reply

SHOW 2 REPLIES

meganlee ★ 1083 September 13, 2018 9:31 AM

Translated into Java version

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
```

Read More

8 Share Reply

daxios ★ 9 September 21, 2018 7:20 PM Report

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 ?

6 Share Reply

SHOW 2 REPLIES

kyzn ★ 4 January 15, 2020 7:08 AM

Why the 1 index?

4 Share Reply

sushanth ★ 4 June 23, 2018 1:57 AM

class Solution {
public int[] twoSum(int[] numbers, int target) {
HashMap<Integer,Integer> hmap = new HashMap<Integer,Integer>();
int diff = 0;

Read More

4 Share Reply

yasheng ★ 1 April 30, 2020 11:26 PM Report

implementation of two pointers in python

```
def twoSum(self, numbers: List[int], target: int) -> List[int]:
```

Read More

1 Share Reply

lh5712 ★ 1 October 16, 2019 4:54 AM Report

The binary search solution in Python 2:

```
class Solution(object):
    def twoSum(self, numbers, target):
```

Read More

1 Share Reply

SHOW 1 REPLY