Previous (/articles/fraction-recurring-decimal/) Next (/articles/linked-list-cycle/)

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

-ii-input-array-sorted/) (/ratings/107/20/?return=/articles/two-sum-ii-input-array-sorted/) (/ratings/107/20/?return=/articles/two-sum-ii-input-array-sorted/)

Average Rating: 4.70 (20 votes)

March 12, 2016 | 4.8K views

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. Please note that 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.

```
Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2
```

Solution

Approach #1 (Two Pointers) [Accepted]

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.

```
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 $O(1)$.

Analysis written by @maonag6.

Rate this article:
(/ratings/107/20/?return=/articles/two-sum-ii-input-array-sorted/) (/ratings/107/20/?return=/articles/

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

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

Join the conversation

Login to Reply