1. Two Sum

Friday, April 29, 2022 3:56 PM

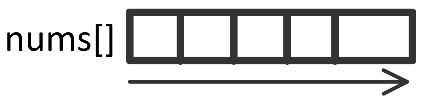
LeetCode 1. Two Sum (easy)

Given an array of integers *nums* and an integer *target*, return indices of the two numbers such that they add up to *target*.
You may assume that each input would have ***exactly* one solution**, and you may not use the same element twice.
You can return the answer in any order.
Example 1:
Input: nums = [2, 7, 11, 15], target = 9
Output: [0, 1]
Example 2:
Input: nums = [3, 2, 4], target = 6
Output: [1, 2]
Example 3:
Input: nums = [3, 3], target = 6
Output: [0, 1]

Solution 1 - brute force O(n^2) quadratic-time
Iterate from i = 0…n-1 through array and for each element find if the complement exists. The complement is the value needed to add to the current element in the array to equal the target value.

complement = target - currentValue

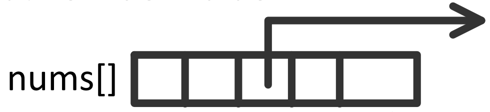
This would require a nested for loop and because we will be searching through n elements for each of the n elements in search for the complement, this would end up being a quadratic-time O(n^2) solution.



complement = target - nums[i]

Scan the nums[] array from left to right and for each element compute the complement and search for that complement sequentially in the nums[] array.

Solution 2 - double pass hashtable O(n) constant-time
We can improve our solution/algorithm by improving the searching/accessing operation time from O(n) linear-time to O(1) constant-time with the help of a hashtable data structure.
As we see each element *and it does not already exist in the hashtable* in the nums[] array, we will take it's index location and the element stored at that index and insert it into our hashtable.
We will then loop through the nums[] array and check if the complement exists in our hashtable and its relative index value.



InsertInHashTable(element, index) —>

complement = target - nums[i]

1. Scan through the nums[] array and insert each element and its corresponding index location into our hashtable.

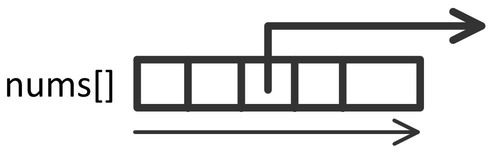
HashTable

Element	Index
2	0
7	1
11	2
15	3

2. Now we will again loop through nums[] array and check if the corresponding complement element is stored in our hashtable.

This algorithm demonstrates traversing the nums[] and manipulating the hashtable twice: to first insert all elements & corresponding indices into the hashtable & then loop through nums[] array again to check if the corresponding complement exists in our hashtable. Although insertion/access operations are O(1) constant-time, the overall time complexity would be O(n) because we will have to loop through the entire nums[] array to "set-up" our hashtable.

Solution 3 - single pass hashtable O(n) linear-time
We can improve our hashtable implementation by only doing a single pass through the nums[] array. We will add each element and its relative index position and check if the complement already exists in the hashtable… This will replicate a single-pass hashtable implementation instead of adding all values to the hashtable and then search sequentially for each element in nums[] if its complement exists in the hashtable.



InsertInHashTable(element, index) &
SearchHashTableFor(complement)

complement = target - nums[i]

1. Scan through the nums[] array and insert each element and its corresponding index location into our hashtable.

HashTable

Element	Index
2	0
7	1
11	2
15	3

This algorithm demonstrates traversing the nums[] and manipulating the hashtable only once: we will loop through the nums[] array once and insert each element and its corresponding index into our hashtable **AND** we will also simultaneously check if each element's complement exists in the hashtable. Although insertion/access operations are O(1) constant-time, the overall time complexity would be O(n) because we will have to loop through the entire nums[] array to "set-up" our hashtable. This is better than ***Solution 2*** because we only iterate through nums[] array only once (at most n times).