**1.Two Sum**

**（1）Question**

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**（2）Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

**Quick Navigation**

* [Solution](https://leetcode.com/articles/two-sum/#solution)
  + [Approach #1 (Brute Force) [Accepted]](https://leetcode.com/articles/two-sum/#approach-1-brute-force-accepted)
  + [Approach #2 (Two-pass Hash Table) [Accepted]](https://leetcode.com/articles/two-sum/#approach-2-two-pass-hash-table-accepted)
  + [Approach #3 (One-pass Hash Table) [Accepted]](https://leetcode.com/articles/two-sum/#approach-3-one-pass-hash-table-accepted)

**(3)Solution**

**Approach #1 (Brute Force) [Accepted]**

The brute force approach is simple. Loop through each element xxx and find if there is another value that equals to target−xtarget - xtarget−x.

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

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**for** **(int** j **=** i **+** 1**;** j **<** nums**.**length**;** j**++)** **{**

**if** **(**nums**[**j**]** **==** target **-** nums**[**i**])** **{**

**return** **new** **int[]** **{** i**,** j **};**

**}**

**}**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

**Complexity Analysis**

* Time complexity : O(n2)O(n^2)O(n​2​​). For each element, we try to find its complement by looping through the rest of array which takes O(n)O(n)O(n) time. Therefore, the time complexity is O(n2)O(n^2)O(n​2​​).
* Space complexity : O(1)O(1)O(1).

**Approach #2 (Two-pass Hash Table) [Accepted]**

To improve our run time complexity, we need a more efficient way to check if the complement exists in the array. If the complement exists, we need to look up its index. What is the best way to maintain a mapping of each element in the array to its index? A hash table.

We reduce the look up time from O(n)O(n)O(n) to O(1)O(1)O(1) by trading space for speed. A hash table is built exactly for this purpose, it supports fast look up in *near* constant time. I say "near" because if a collision occurred, a look up could degenerate to O(n)O(n)O(n) time. But look up in hash table should be amortized O(1)O(1)O(1) time as long as the hash function was chosen carefully.

A simple implementation uses two iterations. In the first iteration, we add each element's value and its index to the table. Then, in the second iteration we check if each element's complement (target−nums[i]target - nums[i]target−nums[i]) exists in the table. Beware that the complement must not be nums[i]nums[i]nums[i] itself!

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

Map**<**Integer**,** Integer**>** map **=** **new** HashMap**<>();**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

map**.**put**(**nums**[**i**],** i**);**

**}**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**int** complement **=** target **-** nums**[**i**];**

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

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

**}**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

**Complexity Analysis:**

* Time complexity : O(n)O(n)O(n). We traverse the list containing nnn elements exactly twice. Since the hash table reduces the look up time to O(1)O(1)O(1), the time complexity is O(n)O(n)O(n).
* Space complexity : O(n)O(n)O(n). The extra space required depends on the number of items stored in the hash table, which stores exactly nnn elements.

**Approach #3 (One-pass Hash Table) [Accepted]**

It turns out we can do it in one-pass. While we iterate and inserting elements into the table, we also look back to check if current element's complement already exists in the table. If it exists, we have found a solution and return immediately.

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

Map**<**Integer**,** Integer**>** map **=** **new** HashMap**<>();**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**int** complement **=** target **-** nums**[**i**];**

**if** **(**map**.**containsKey**(**complement**))** **{**

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

**}**

map**.**put**(**nums**[**i**],** i**);**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

**Complexity Analysis:**

* Time complexity : O(n)O(n)O(n). We traverse the list containing nnn elements only once. Each look up in the table costs only O(1)O(1)O(1) time.
* Space complexity : O(n)O(n)O(n). The extra space required depends on the number of items stored in the hash table, which stores at most nnn elements.