1. Two Sum

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]
Explanation: Because nums[0] + nums[1] == 9, we return [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]

Constraints:

- `2 <= nums.length <= 10^4`
- `-10^9 <= nums[i] <= 10^9`
- `-10^9 <= target <= 10^9`
- **Only one valid answer exists.**

here was my Engrish explanation of how i solved it

its for each number in the array we index it then we start with the first value (firstvalue_x) then subtract that from the target value(target_val), if the result (secondvalue_y) is in the index and isnt the index number(firstvalue_x) we subtracted from the target value(target_val) that is the answer(answer) otherwise continue with this loop by increasing the (firstvalue_x). we dont have to look at values we have tested already

```
public class Solution {
    public int[] TwoSum(int[] nums, int target) {
        Dictionary<int, int> dict = new Dictionary<int, int>();
 // We create an index (dictionary) to store each number and its position in the array.
        for (int n = 0; n < nums.Length; n++) {
// We start a loop to go through each number in the array. The variable 'i' is the position of the current number (firstvalue_x).
            int complement = target - nums[n];
// We subtract the current number (firstvalue_x) from the target value (target_val), the result is the complement (secondvalue_y).
            if (dict.ContainsKey(complement) && dict[complement] != n) {
// We check if the complement (secondvalue_y) is in the index (dictionary) and its position is not the same as the current position (firstvalue_x).
                return new int[] { dict[complement], n };
 // If the above condition is true, we've found the answer (answer). We return the positions of the two numbers that add up to the target.
            }
            dict[nums[n]] = n;
// If the complement is not in the index, we add the current number and its position to the index (dictionary) and continue with the loop by increasing the current position (firstvalue_x).
        }
                throw new Exception("There's a problem with your code or the input array is bad");
// If we've gone through all the numbers and haven't found two numbers that add up to the target, we throw an exception.
// we need this to allow compiler to compile because the compiler requires that all code paths in a function must return a value if the function is declared to return a value
// we *have* to have this or it just fails at compiling
    }
}
```

cases for question

```
case 1 [2,7,11,15] target = 9
case 2 [3,2,4] target = 6
case 3 [3,3] target = 6
```

The code works for these test questions
Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?
i asked chat gpt coz idk if it is off the top of my head coz i cant think lol

"Yes, the provided solution fits the follow-up question. The time complexity of this solution is O(n), which is less than O(n^2).

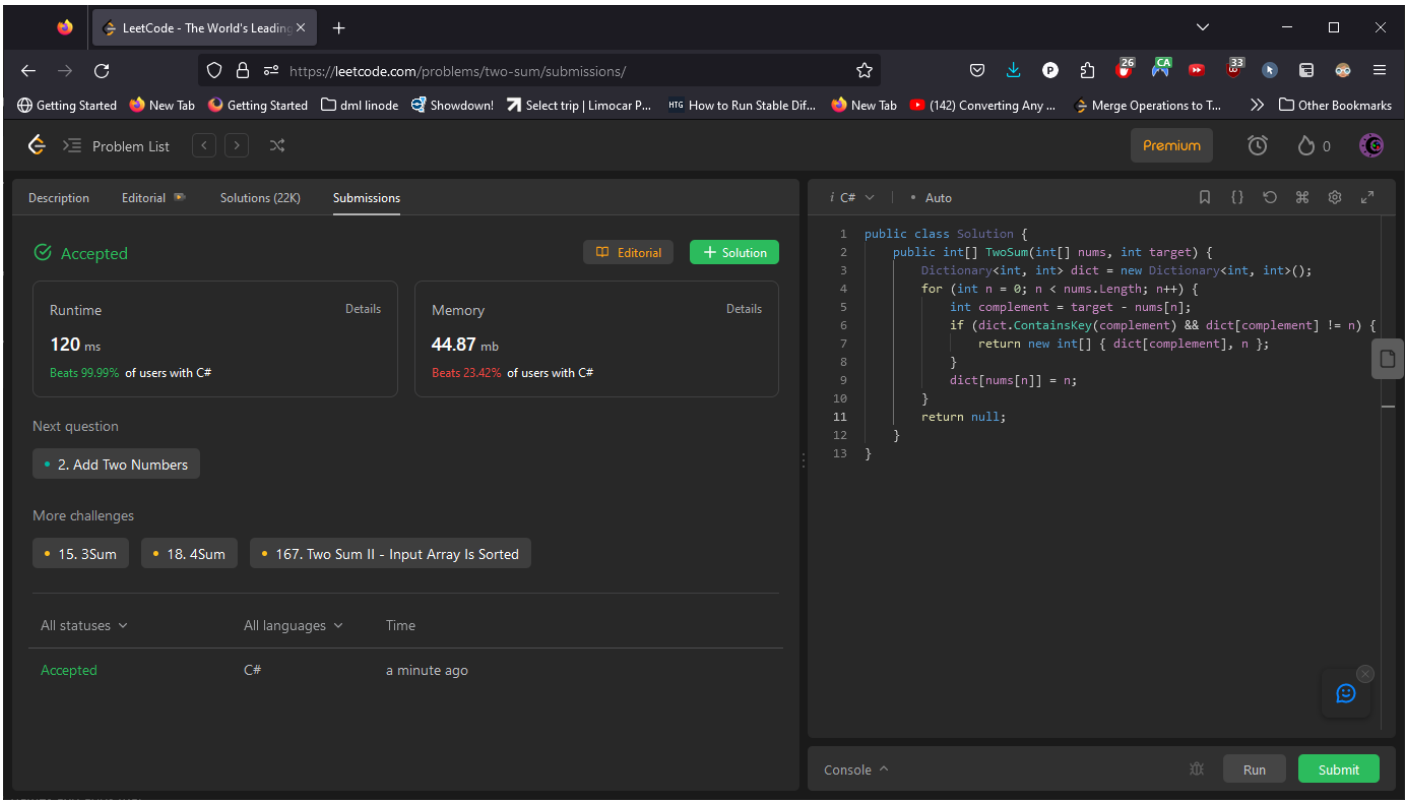
Here's why:

The solution iterates over the `nums` array only once. For each number, it performs operations that take constant time: calculating the complement, checking if the complement is in the dictionary, and adding a number to the dictionary. Therefore, the time complexity is proportional to the size of the `nums` array, which is O(n)."

what does leetcode say to my answer?
well it likes it :D i guess its about as fast as it can be, especially if you remove the comments

---

i did some minor optimizations
and heres what i got



Intuition

The problem asks for finding two numbers in an array that add up to a specific target. The "easy" but inefficient way would be to check all possible pairs of numbers, which would take O(n^2) time. Instead, we can use the property of addition: for any number z, if x + y equals z, then y must be equal to z minus x. So, for each number in the array, we can check its complement to the target [z]

(i.e., [z] - x) and then check if this complement is in the array. If it is, and it's not the same index we're currently looking at, we've found the two numbers we need. To make this check fast, we can use a hash table (or a Dictionary in C#) to store the numbers from the array and their indices.

Approach

My English pseudo code/thought process is
"for each number in the array, we index it.
then we start with the first indexed value (firstvalue_x)
then subtract that from the target value (target_val_z),
if the result (secondvalue_y) is in the index and isnt the index number(firstvalue_x) we subtracted from the target value(target_val_z) that is the answer ([x, y]) otherwise continue with this loop by increasing the (firstvalue_x).
we do not have to look at values we have tested already because that would be redundant
The approach is to iterate through the array and for each number, calculate its complement to the target and check if this complement is already in the Dictionary. If it is, and it's not the same element (i.e., it has a different index), we've found two numbers that add up to the target and we return their indices. If the complement is not in the Dictionary, we add the current number and its index to the Dictionary. This way, we don't have to revisit numbers we've already processed, which would be redundant.

Complexity

- Time complexity:
  The time complexity is O(n), where n is the length of the array. We're iterating through the array once, and for each number, we're performing constant time operations (calculating the complement, checking if it's in the dictionary, and adding a number to the dictionary).
- Space complexity:
  The space complexity is also O(n), where n is the length of the array. In the worst case, we might end up adding every number in the array to the dictionary.

Extra

this is my first post.
It took me a bit of digging to learn what to use in c# for this specifically coz ive only ever used it for unity @~@ and i usuially use python so it was a bit of a challenge to get the code to not give an error XD

and i found out that it wont compile if we dont have an exception handler if theres an error in a hypothetical input....
sooo since we are using c# ive found that using

```
throw new Exception("exception");
```

is slower on avg
so

```
return null;
```

is used
also since im new to leetcode.com ive foundout that the compieler speeds can be subject to rng, also submissions have processing priority ig.
my codes runtime is 120 and worst is 145
its avg is in the low 130s

Code

```
public class Solution {
    public int[] TwoSum(int[] nums, int target) {
        Dictionary<int, int> dict = new Dictionary<int, int>();
        for (int n = 0; n < nums.Length; n++) {
            int complement = target - nums[n];
            if (dict.ContainsKey(complement) && dict[complement] != n) {
                return new int[] { dict[complement], n };
            }
            dict[nums[n]] = n;
        }
        return null;
    }
}
```

Revision #2
Created 28 July 2023 13:52:19 by naruzkurai
Updated 28 July 2023 16:57:06 by naruzkurai