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:

$$2 \leq nums.length \leq 10^4$$

$$-10^9 \leq nums[i] \leq 10^9$$

$$-10^9 \leq target \leq 10^9$$

Only one valid answer exists.

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

Brute Force Method

Simply start from the first index and comparing every single other indices to see if the combination is equivalence to the target

$Runtime : O(n^2)$

```
class Solution(object):
  def twoSum(self, nums, target):
      for a in range(0,len(nums)):
          current = nums[a]
          for b in range(1,len(nums)):
              if (current + nums[b]) == target:
                  return [a,b]
       return 0
```

Optimal Solution

If we take the target and minus the a particular index, we would get the `difference` (the other value we are looking for). Run through the array and store the previous values in the `hash map` as we go. If our current index yield a differences equal to the value that is already stored in the map. We got a match and return the pair. Doing so will allow us to pass the array only once

$Runtime : O(n)$

```
class Solution(object):
    def twoSum(self, nums, target):
        prevMap = {} # val:index
        for i,n in enumerate(nums):
            diff = target - n
            if diff in prevMap:
                return [prevMap[diff],i]
            prevMap[n] = i
        return 0
```