Two Sum

Difficulty	Easy
Category	Arrays
Question	https://leetcode.com/problems/two-sum/
Solution	https://youtu.be/KLlXCFG5TnA
Status	Done

Question

> 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

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]
```

Approach 1: Brute Force

I used 2 for loops for my first solution.
The time complexity was: O(n^2)
Space complexity was: O(n)

```
Brute Force:
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        list1=[]
        for i in range(0,len(nums)):
            for j in range(i+1,len(nums)):
                if target==nums[i]+nums[j]:
                    list1.append(i)
                    list1.append(j)
                    return list1
        return list1
```

Approach 2: Hash Table

I used a hash table to improve my previous approach. For example, my array was [1,2,3,4,5] and my target sum was 9. The resulting sum would be 9-1=8. So I would start with 1 and start iterating over every element one by one and check if 8 is in my hash table. If it is just return 1,8 else store the number in the dictionary and keep on iterating until you find the required sum.

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dict1={}
```

```
        for i,num in enumerate(nums):
            expected_diff=target-num
            if expected_diff in dict1:
                return [j,dict1[nums]]
            else:
                dict1[nums]=i

        return []
```

Time complexity: O(n)

Because you are traversing the entire array. And the rest operations are constant time operations. So it just takes linear time complexity

Space Complexity: O(n)

Since you are storing the element in the dictionary

Approach 3: Using left and right pointers

In this approach we will sort the array first. Then we define two pointers left and right. While the left<right:

Now we calculate the sum of left and right pointer. If the difference is less than the target then we increase the left pointer else we decrease the right pointer. When we get the desired sum, we return our numbers:

```
from typing import List


class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        left_pointer=0
        right_pointer=len(nums)-1
        nums_copy=nums.copy()
        nums_copy.sort()

        while(left_pointer<right_pointer):
            sum_value=nums_copy[left_pointer]+nums_copy[right_pointer]
            if target == sum_value:
                left_index=  nums.index(nums_copy[left_pointer])
                right_index= len(nums)-nums[::-1].index(nums_copy[right_pointer])-1
                return [left_index,right_index]
            elif sum_value < target:
                left_pointer+=1
```

```
            elif sum_value > target:
                right_pointer-=1

        return []
```

Time complexity: O(nlogn)

Because you are traversing the entire array. And the rest operations are constant time operations. So it just takes linear time complexity

Space Complexity: O(1)

Since you are not storing anything