LeetCode 1 [Two Sum]

Given an array of integers nums and an integer target, return indices of
the two numpers 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 your answer in any order.

$$\text{nums} = [\overset{0}{15}, \overset{1}{\underline{7}}, \overset{2}{\underline{2}}, \overset{3}{11}] \qquad \text{target} = 9$$

[1, 2]
[2, 1]

Brute Force

[15, 7, 2, 11]   t: 9

We skip 15
Then 9 - 7 = 2, now it's necessary to check if 2 is present.

Sorting

[15, 7, 2, 11]   t: 9
[2, 7, 11, 15]   9 - 2 = 7
We will look for 7

B.S. (log n)

We can improve using and additiontal D.S.

$$[\overset{0}{1}, \overset{1}{7}, \overset{2}{3}, \overset{3}{2}] \qquad t = 9$$

↑ ↑
2

H.M.

key	value
1	0
7	1
3	2

Answer ⟶ [3, 1]

Time: O(n)
Space: O(n)

Java Solution

```
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);
    }

    return new int[]{-1,-1};
}
```

Kotlin solution

```
fun twoSum(nums: IntArray, target: Int): IntArray {
    val map = HashMap<Int, Int>()

    For(i in nums.indices){
        val complement = target - nums[i]

        if(map.containsKey(complement)) {
            return intArrayOf(map[complement]!!, i)
        }
        map.put(nums[i], i)
    }

    return intArrayOf(-1, -1)
}
```