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.**

Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

1. Two Sum

```
/*
   Brute force
   Time complexity: O(n^2)
   Space complexity: O(1)
*/
typedef std::vector<int> vi;

class Solution {
   public:
      vi twoSum(vi& nums, int target) {
         int n=nums.size();

         for(int i=0;i<n-1;++i){
            for(int j=i+1;j<n;++j){
               if(nums[i]+nums[j]==target) return {i,j};
            }
         }
         return {};
      }
};
```

1. Two Sum

```
/*
   Hashing+Math
   Time complexity: O(n)
   Space complexity: O(n)
*/
typedef std::vector<int> vi;

class Solution {
   public:
      vi twoSum(vi& nums, int target) {
         int n=nums.size();

         std::unordered_map<int,int> seen;

         for(int i=0;i<n;++i){
            if(seen.find(nums[i])!=seen.end()) return {seen[nums[i]],i};
            seen[target-nums[i]]=i;
         }

         return {};
      }
};
```