Two Sum (Easy)

Given an array of integers `nums` and an integer `target`, return the indices `i` and `j` such that `nums[i] + nums[j] == target` and `i != j`.

You may assume that *every* input has exactly one pair of indices `i` and `j` that satisfy the condition.

Return the answer with the smaller index first.

Example 1:

```
Input:
nums = [3,4,5,6], target = 7

Output: [0,1]
```

Explanation: `nums[0] + nums[1] == 7`, so we return `[0, 1]`.

Example 2:

```
Input: nums = [4,5,6], target = 10

Output: [0,2]
```

Example 3:

```
Input: nums = [5,5], target = 10

Output: [0,1]
```

Constraints:

- `2 <= nums.length <= 1000`
- `10,000,000 <= nums[i] <= 10,000,000`
- `10,000,000 <= target <= 10,000,000`

思路:

1.暴力比對T:O(N^2), S:O(1)
2.另建vector pair排序後比對T:O(NlogN), S:O(n)

3.hash比對T:O(NlogN), S:O(n)

作法二

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<pair<int,int>> a;
        for(int i=0;i<nums.size();i++){
            a.push_back({nums[i],i});
        }
        sort(a.begin(),a.end());
        int i=0,j=nums.size()-1;
        while(i<j){
            int sum = a[i].first+a[j].first;
            if(sum == target){
                return {min(a[i].second,a[j].second),max(a[i].second,a[j].second)};
            }
            else if(sum<target){
                i++;
            }
            else{
                j--;
            }
        }
        return {};
    }
};
```

作法三

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> index;   //val→index

        for(int i=0;i<nums.size();i++){
            if(index.count(target - nums[i])){
                return {index[target - nums[i]], i};
            }
            index[nums[i]]=i;
        }


    }
};
```