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?

Approach:

The code implements the two-pointer technique using a HashMap to efficiently find two indices in the array `nums` whose elements sum up to `target`.

Code:

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int n=nums.length;
        Map<Integer,Integer> map=new HashMap<>();
        int[] result=new int[2];
        for(int i=0;i<n;i++){
            if(map.containsKey(target-nums[i])){
                result[1]=i;
                result[0]=map.get(target-nums[i]);
                return result;
            }
            map.put(nums[i],i);
        }
        return result;
    }
}
```