Two Sum

Given an array of integers nums and an integer target, return indicesof 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 <= 104`
- `-109 <= nums[i] <= 109`
- `-109 <= target <= 109`
- `Only one valid answer exists.`

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

Algorithm:

- Create a map of int to int storing the number and its location in the array
- Loop through the array
- Find for the complement of the number in the map
- If the complement is found, push the indices of the pair into the array
- If not found then add the integer at ith position into the map
- Return the result array

Code:

```
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
      vector<int> twoSum(vector<int>& nums, int target)
            {
          map<int,int> m;
          vector<int> v;
          int n= nums.size();
          for(int i=0;i<n;i++)
          {

               int diff = target - nums[i];
               if(m.find(diff) != m.end())
               {
               auto p = m.find(diff);
               v.push_back(p->second);
               v.push_back(i);
               }
               m.insert(make_pair(nums[i],i));
                    }

                            return v;
        }
};


int main()
{
    Solution s ;
    vector<int> v;
    v={1,6,3,2,5};
    vector<int> result= s.twoSum (v, 11);
    for(int i: result)
    {
        cout<<i<<endl;
    }
        return 0;

}
```