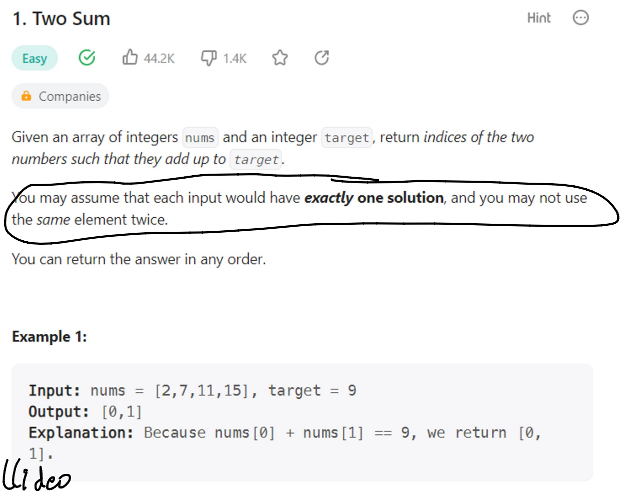
19. Two Sum



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.

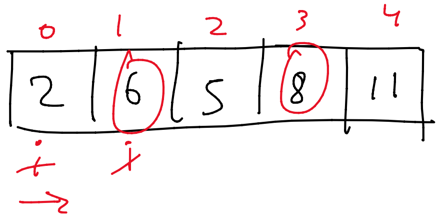
You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0,
1].
```

Old Video

1) Brute force



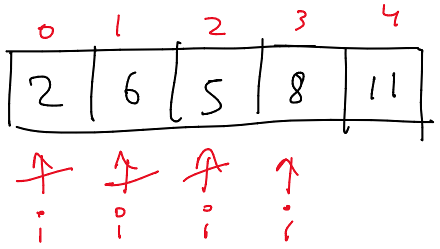
target = 14

14-2=12 → check if this exist in the array or not.

14-6 = 8 → this exist in array

T.C ⇒ $O(n^2)$ S.C ⇒ $O(1)$

2) Using hashtable



target = 14

Check if it's there in hash table or not

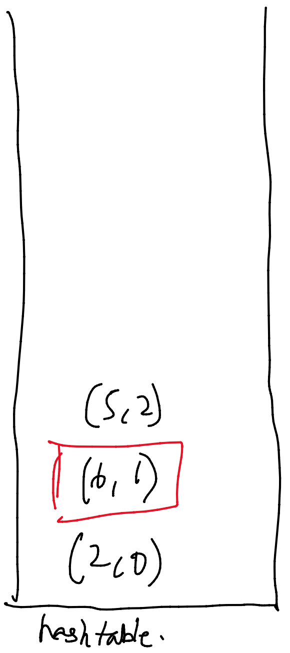
(if not there then push it in hash table the array)

14- 2 = 12

14 - 6 = 8

14 - 5 = 9

14 - 8 = 6 → this exist in index '1'.



(1, 3) ⇒ o/p

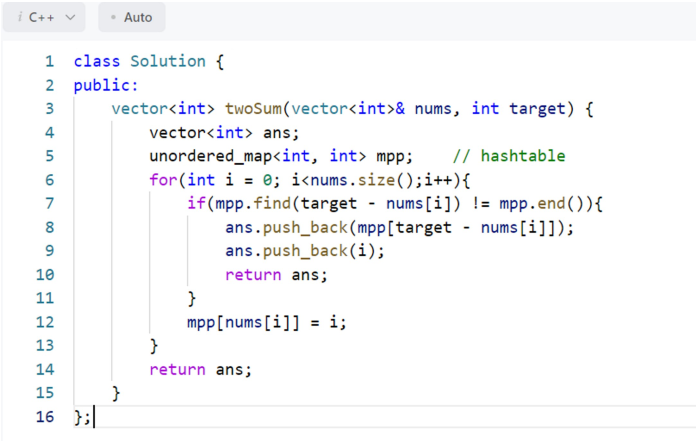
→ from the pointer i.

→ from the hashtable

T.C ⇒ $O(n)$

S.C ⇒ O(N) //Storing value in hashtable

So in C++, we use unordered map



```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<int> ans;
        unordered_map<int, int> mpp;    // hashtable
        for(int i = 0; i<nums.size();i++){
            if(mpp.find(target - nums[i]) != mpp.end()){
                ans.push_back(mpp[target - nums[i]]);
                ans.push_back(i);
                return ans;
            }
            mpp[nums[i]] = i;
        }
        return ans;
    }
};
```

New video

arr[] = { 2, 6, 5, 8, 11 }   target = 14

1st part ⇒ Yes / No   2nd part ⇒ return the index of two element.

Brute force

```
for (i=0 → n)
{
  for (j = i+1 → n)
  {
    if(i == j) continue; //Because we cannot pick up same elements
    if (arr[i] + arr[j] == target)
    {
       print(yes) // 1st part
       print{i, j} // 2nd part
    }
  }
```

}

}

T C ⇒ $O(n^2)$ S. C ⇒ $O(1)$

2nd Approach

Hashing

arr[] = { 2, 6, 5, 8, 11 }   with indices 0 1 2 3 4   target = 14

Check if its there in hash table or not

(if not there then push it in hash table the array)

$14 - 2 = 12$

$14 - 6 = 8$

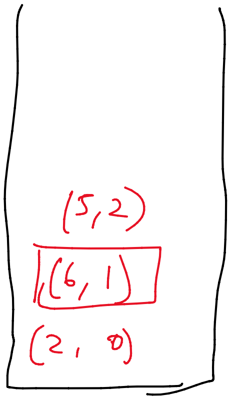
$14 - 5 = 9$

$\boxed{14 - 8 = 6}$ → there in hashmap

[3, 1]

→ from hashmap.

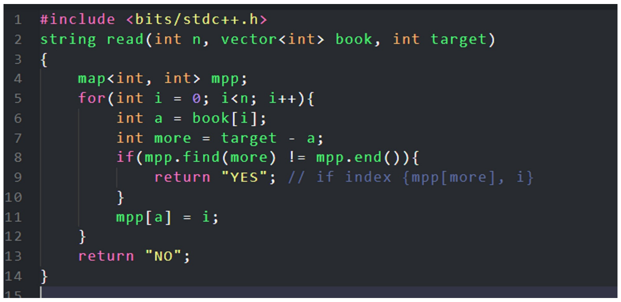
→ from i pointer



Hash map

(ele, index)
↓ key   ↓ Value

T. C ⇒ O(N logN)   S. C ⇒ O(N)



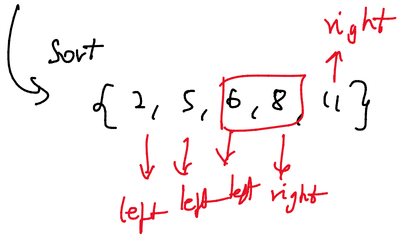
```
#include <bits/stdc++.h>
string read(int n, vector<int> book, int target)
{
    map<int, int> mpp;
    for(int i = 0; i<n; i++){
        int a = book[i];
        int more = target - a;
        if(mpp.find(more) != mpp.end()){
            return "YES"; // if index {mpp[more], i}
        }
        mpp[a] = i;
    }
    return "NO";
}
```

3rd Approach

Without using map.

Using two pointer appro

arr[] = { 2, 6, 5, 8, 11 }   target = 14 arr[] = { 2, 6, 5, 8, 11 }   target = 14



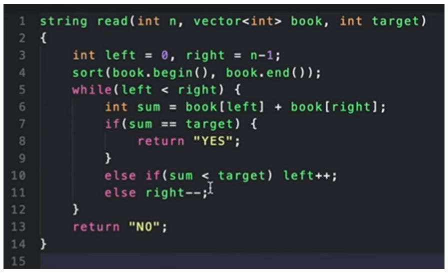
$2 + 11 = 13 < 14$ (So move left)

$5 + 11 = 16 > 14$ (move right)

$5 + 8 = 13 < 14$ (move left)

$6 + 8 = 14$ ✓

It solves part-1, but for part-2 (index) You need to put arr[] in another d.s. {(2,0) (6,1) (5,2) (8,3) (11,4)} alog with index and sort it. for part-2 this is not best.



```cpp
string read(int n, vector<int> book, int target)
{
    int left = 0, right = n-1;
    sort(book.begin(), book.end());
    while(left < right) {
        int sum = book[left] + book[right];
        if(sum == target) {
            return "YES";
        }
        else if(sum < target) left++;
        else right--;
    }
    return "NO";
}
```

T.C ⇒ $O(n) + O(n\log n)$

S.C ⇒ $O(1)$