LEETCODE - 1

Method 1 – Brute Force ($O(n^2)$)

```
vector<int> twoSum(vector<int>& nums, int target) {
    vector<int> ans;
    for(int i=0;i<=nums.size()-2;i++){
        for(int j=i+1; j<nums.size(); j++){
            if(nums[i]+nums[j]==target){
                ans.push_back(i);
                ans.push_back(j);
            }
        }
    }
    return ans;
}
```

Logic Flow

1. **Outer loop (`i`)** – 0th element se lekar 2nd last element tak iterate karega.
2. **Inner loop (`j`)** – Har `i` ke liye, uske agle element se last tak iterate karega.
3. **Check sum** – Agar `nums[i] + nums[j] == target`,
   - `ans` me `i` aur `j` push karega.
4. **Return result** – Loop ke baad `ans` return ho jayega.

✅ **Simple but slow** – Har possible pair check hota hai, isliye **$O(n^2)$** time lagta hai.

Method 2 – Hash Map (O(n))

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int,int>mp;
    vector<int>ans;
    for(int i=0; i<nums.size(); i++){
        if(mp.find(target-nums[i])!=mp.end()){
            //if found then this block would be hit
            ans.push_back(mp[target-nums[i]]);
            ans.push_back(i);
        }
        else{
            mp[nums[i]]=i;
        }
    }
    return ans;
}
```

Logic Flow

1. **Hash map** `mp` – Ye element → index store karega.
   - Key = `nums[i]`
   - Value = uska index `i`
2. **Loop through** `nums` – Har element ke liye:
   - **Check complement** = `target - nums[i]`
   - Agar wo complement already `mp` me hai → iska matlab ek earlier element mil gaya jo current element ke saath milkar `target` banata hai.
   - **Push indices**: Pehle stored index (`mp[complement]`) aur fir current index (`i`).
3. **Else block** – Agar complement nahi mila, to current element ko map me store kar lo (`mp[nums[i]] = i`).
4. End me `ans` return ho jayega.

✅ **Faster** – Sirf **O(n)** time lagta hai kyunki har element ek hi baar process hota hai aur hash lookup average O(1) hota hai.

📌 **Difference Summary**

Method	Time Complexity	Space Complexity	Approach
Brute Force	$O(n^2)$	O(1)	Har pair check karna
Hash Map	O(n)	O(n)	Complement ko map me store karke check karna

---

Code

```
vector<int> twoSum(vector<int>& nums, int target) {
  unordered_map<int,int>mp;
  vector<int>ans;
  for(int i=0; i<nums.size(); i++){
      if(mp.find(target-nums[i])!=mp.end()){
         ans.push_back(mp[target-nums[i]]);
         ans.push_back(i);
      }
      else{
         mp[nums[i]]=i;
      }
  }
  return ans;
}
```

Example

```
nums = [2, 7, 11, 15]
```

```
target = 9
```

Step-by-step Dry Run

Initial State

- `mp` = {} (empty)
- `ans` = {} (empty)

---

i = 0

- nums[0] = **2**
- complement = `target - nums[0]` = `9 - 2` = **7**
- **Check**: Is 7 in `mp`? ❌ No
- Store → `mp[2] = 0`
- **mp** now = `{ 2 → 0 }`
- **ans** = {}

---

i = 1

- nums[1] = **7**
- complement = `9 - 7` = **2**
- **Check**: Is 2 in `mp`? ✅ Yes
- Push → `ans.push_back(mp[2])` → ans = `[0]`
- Push → `ans.push_back(1)` → ans = `[0, 1]`
- **mp** stays = `{ 2 → 0 }`
- **ans** = `[0, 1]`

---

i = 2 and **i = 3** ka process logically chalega, but yaha pe answer already mil chuka hai, loop end hone ke baad `ans` return ho jayega.

Final Output

```
[0, 1]
```

Visualization Table

i	nums[i]	complement = target - nums[i]	Found in mp?	Action	mp after step	ans after step
0	2	7	No	store	{2:0}	{}
1	7	2	Yes	push	{2:0}	{0,1}
2	11	-2	No	store	{2:0, 11:2}	{0,1}
3	15	-6	No	store	{2:0, 11:2, 15:3}	{0,1}