

3.

Two Sum

1

0

EASY

■ What You Will Learn

- ✓ Brute force vs HashMap approach
- ✓ The complement trick: if $a+b=\text{target}$, then $b=\text{target}-a$
- ✓ How HashMap gives $O(n)$ over $O(n^2)$ brute force

■ Intuition First

You need two numbers that add up to a target. For each number you see, ask: 'Have I already seen the number that would complete this pair?' If yes — found it. If no — remember this number for future pairs.

■ HashMap Approach

- For each $\text{arr}[i]$, compute $\text{complement} = \text{target} - \text{arr}[i]$
- Check if complement exists in HashMap
- If yes: return $\{\text{map.get}(\text{complement}), i\}$
- If no: store $\text{arr}[i] \rightarrow i$ in map and continue

■ Visual Trace on [2,7,11,15], target=9

Find two numbers summing to 9

2	7	11	15
---	---	----	----

$i=0$: $\text{arr}[0]=2$, $\text{complement}=9-2=7$. Map is empty $\rightarrow$ store $\{2:0\}$

$i=1$: $\text{arr}[1]=7$, $\text{complement}=9-7=2$. Found 2 in map at index 0!

$\rightarrow$ Return $[0, 1]$ ✓

Pair found: index 0 and 1

2	7	11	15
---	---	----	----

■ Java Code

```
public static int[] twoSum(int[] arr, int target) {  
    Map<Integer, Integer> map = new HashMap<>();  
    for (int i = 0; i < arr.length; i++) {  
        int complement = target - arr[i];
```

```
if (map.containsKey(complement)) {  
    return new int[]{map.get(complement), i};  
}  
map.put(arr[i], i); // Store current element  
}  
return new int[]{}; // No solution found  
}
```

■ Complexity

Case	Time	Space	What happens?
Brute	$O(n^2)$	$O(1)$	Check every pair — 2 nested loops
HashMap	$O(n)$	$O(n)$	One pass — trade space for speed

■ Common Mistakes

- Don't use the same element twice — store AFTER checking, not before
- For sorted array: two pointer approach gives $O(n)$ time AND $O(1)$ space

■ Sorted Array? Use Two Pointers

- left=0, right=n-1
- If $arr[left] + arr[right] == target \rightarrow$ done
- If $sum < target \rightarrow left++$
- If $sum > target \rightarrow right--$
- $O(n)$ time, $O(1)$ space — better than HashMap when sorted

■ Practice Problems

[Easy] Find if any pair sums to target (return true/false only)

[Med] Three Sum — find triplets summing to zero

[Med] Two Sum with sorted array using two pointers

■ Quick Revision

- complement = target - arr[i]
- Check complement in HashMap before storing arr[i]

- HashMap: $O(n)$ time, $O(n)$ space
- Sorted array: two pointers, $O(n)$ time, $O(1)$ space
- Store index as value in map, not just existence