# Two Sum

| | |
|---|---|
| # Attempts | 1 |
| Date Solved | @October 21, 2025 |
| Difficulty | Easy |
| Next Review | @October 28, 2025 |
| Status | Solved |
| Topic/Pattern | Array and String Math |

## Link - https://neetcode.io/problems/two-integer-sum?list=neetcode150

## Problem

- Given an array of integers `nums` and an integer `target`, return the indices `[i, j]` such that `nums[i] + nums[j] == target` and `i != j`.
- Exactly **one solution** exists. Return indices with the **smaller index first**.

---

### Examples

| Input | Output | Reason |
|---|---|---|
| `nums = [2, 7, 11, 15], target = 9` | `[0, 1]` | 2 + 7 = 9 |
| `nums = [3, 2, 4], target = 6` | `[1, 2]` | 2 + 4 = 6 |
| `nums = [3, 3], target = 6` | `[0, 1]` | Both 3s sum to 6 |

---

## Approach 1 — Brute Force

- **Idea:** Check all pairs `(i, j)` to see if they sum to `target`.

```
class Solution:
    def twoSum(self, nums: list[int], target: int) -> list[int]:
        n = len(nums)
        for i in range(n):
            for j in range(i+1, n):
                if nums[i] + nums[j] == target:
                    return [i, j]
```

- **Time Complexity:** O($n^2$)
- **Space Complexity:** O(1)
- **Notes:** Simple, slow

---

## Approach 2 — Hash Map / Dictionary

- **Idea:** Store `number → index`. For each number, check if `target - number` exists in the dictionary.

```
class Solution:
    def twoSum(self, nums: list[int], target: int) -> list[int]:
        seen = {}  # number → index
        for i, num in enumerate(nums):
            difference = target - num
            if difference in seen:
                return [seen[difference], i]
            seen[num] = i
```

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)
- **Notes:** Optimal, fast

---

## Edge Cases

- Array with exactly two elements → returns `[0, 1]`
- Negative numbers → works fine
- Duplicate numbers → handled correctly
- Target zero → works

---

## Summary Table

| Approach | Time | Space | Note |
|---|---|---|---|
| Brute Force | $O(n^2)$ | O(1) | Simple |
| Hash Map | O(n) | O(n) | Optimal |