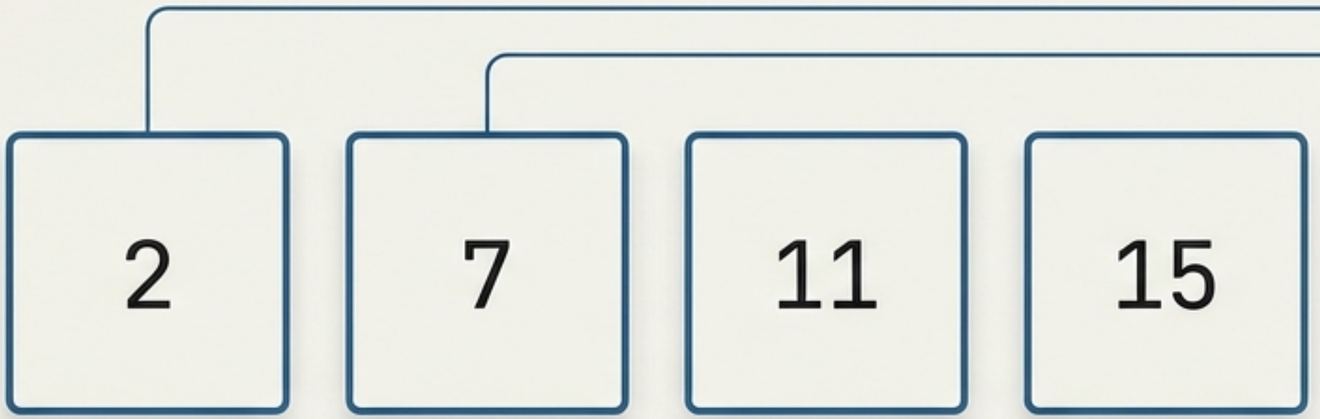
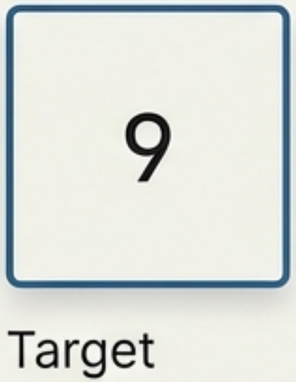
Two Sum: The Art of Algorithmic Trade-offs

Decoding the logic behind the Blind 75's foundational problem.

The Challenge





Constraint: Exactly one solution exists.

Goal: Return indices `[0, 1]`

Reasoning:
`nums[0] + nums[1] == 9`

Strategy A: The Unsorted Input

The Complement Concept

Do not scan for a pair sum ($a + b = target$).

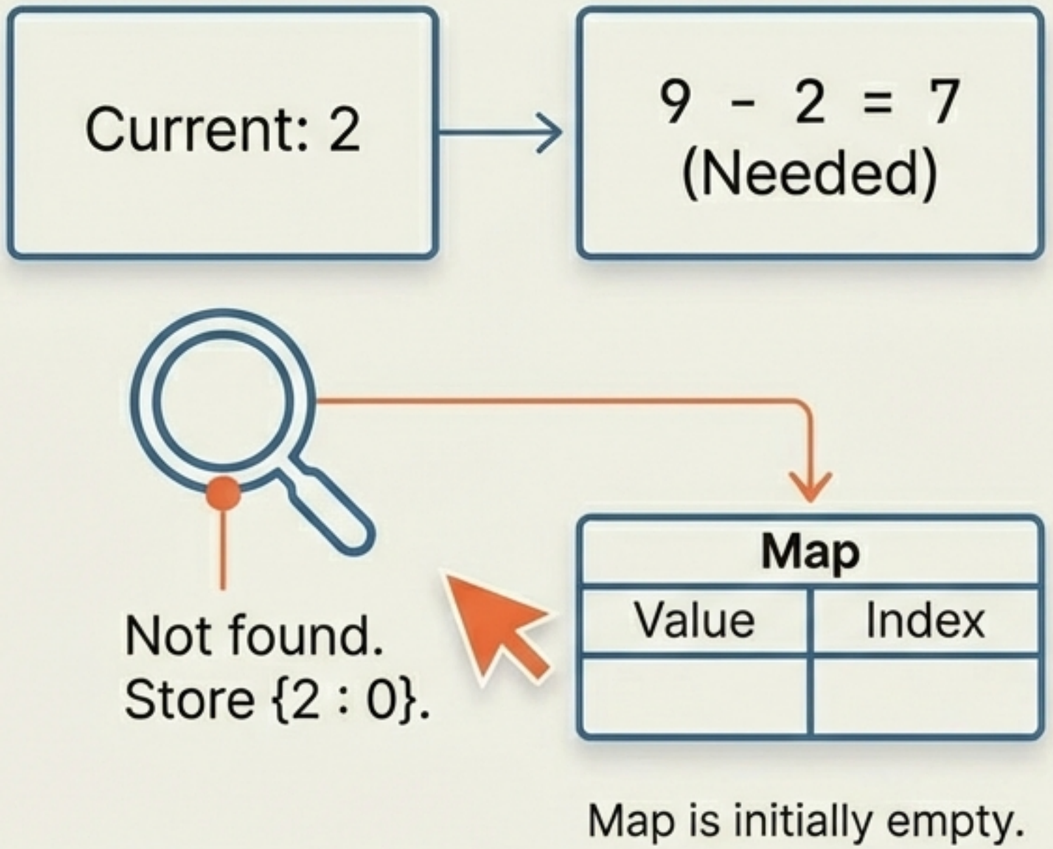
Iterate once and ask: Have I already seen the difference (target – current)?

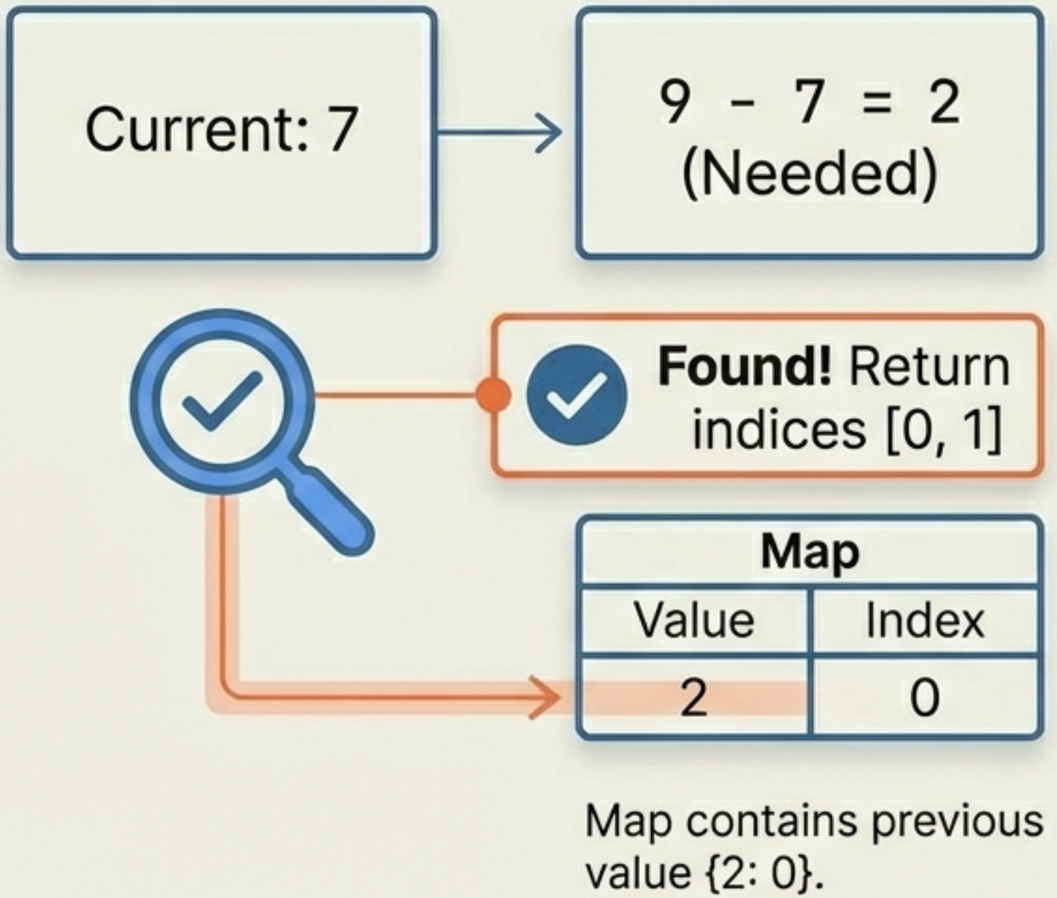
Key (Value Needed)	Value (Index seen)
	5 → 0
	2 → 1
	-2 → 2
...	...

Tool: Hash Map

Remember visited indices with O(1) lookup time.

Visualising the One-Pass Hash Map





Implementation: Trading Space for Time

```python
def twoSum(nums, target):
    prevMap = {}  # val : index

    for i, n in enumerate(nums):
        diff = target - n
        if diff in prevMap:
            return [prevMap[diff], i]
        prevMap[n] = i
```

Access index and value simultaneously

Constant time lookup O(1)

Build memory bank

Analysis: The Cost of Speed

Time Complexity

O(N)

One pass through the list. Map lookups are constant time.

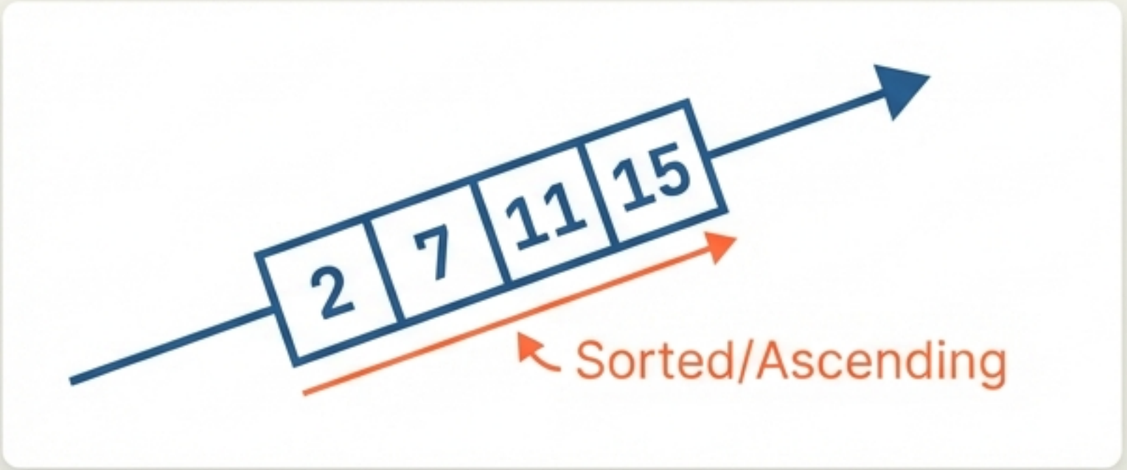
Space Complexity

O(N)

Worst case: store nearly every element in the Map.

Optimal for unsorted data where speed > memory.

The Twist: Sorted Input

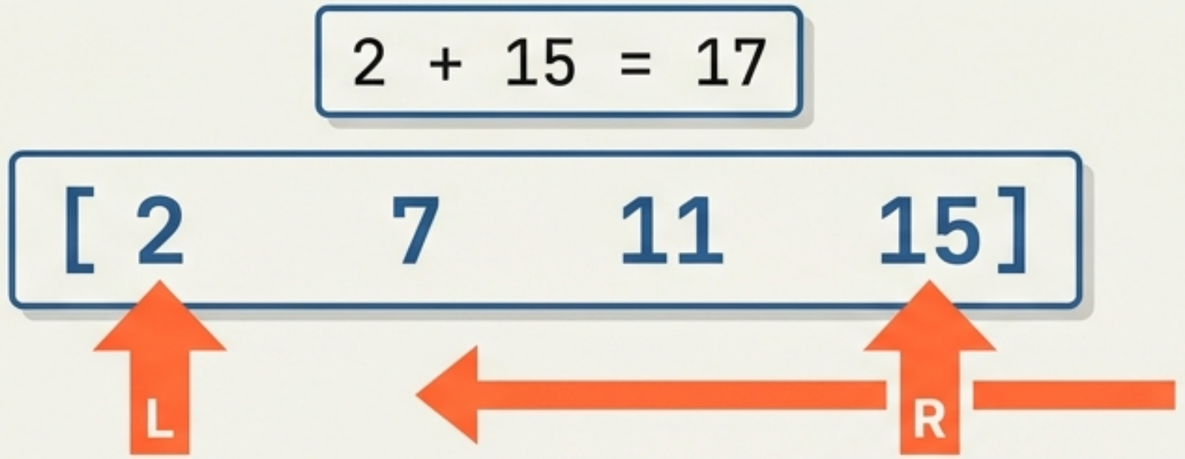


Constraint Change: Input is guaranteed sorted.

Sorted Data + **Directionality** = **No Hash Map Needed**

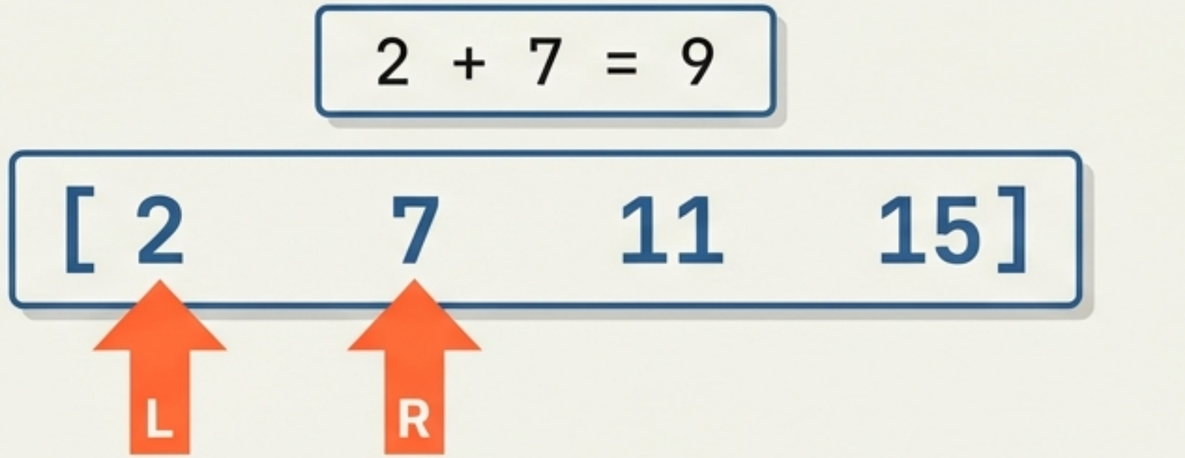
Tool: Two Pointers

Visualising the Pincer Movement



17 > 9 (Target).
Too large.

Move **Right** pointer left



Match Found.

Implementation: The Constant Space Solution in Heay Helvetica Now Display with tight tracking

```
l, r = 0, len(nums) - 1

while l < r:
    curSum = nums[l] + nums[r]
    if curSum > target:
        r -= 1
    elif curSum < target:
        l += 1
    else:
        return [l, r]
```

Termination condition

Sum too big? Decrease right.

Sum too small? Increase left.

The Verdict: Choosing Your Tool

Unsorted Data

Approach	Hash Map (One Pass)
Time	$O(N)$
Space	$O(N)$ ↑
Logic	`target - current`

Sorted Data

Approach	Two Pointers
Time	$O(N)$
Space	$O(1)$ ✓
Logic	`Converge from edges`

Always check input constraints before selecting your data structure.