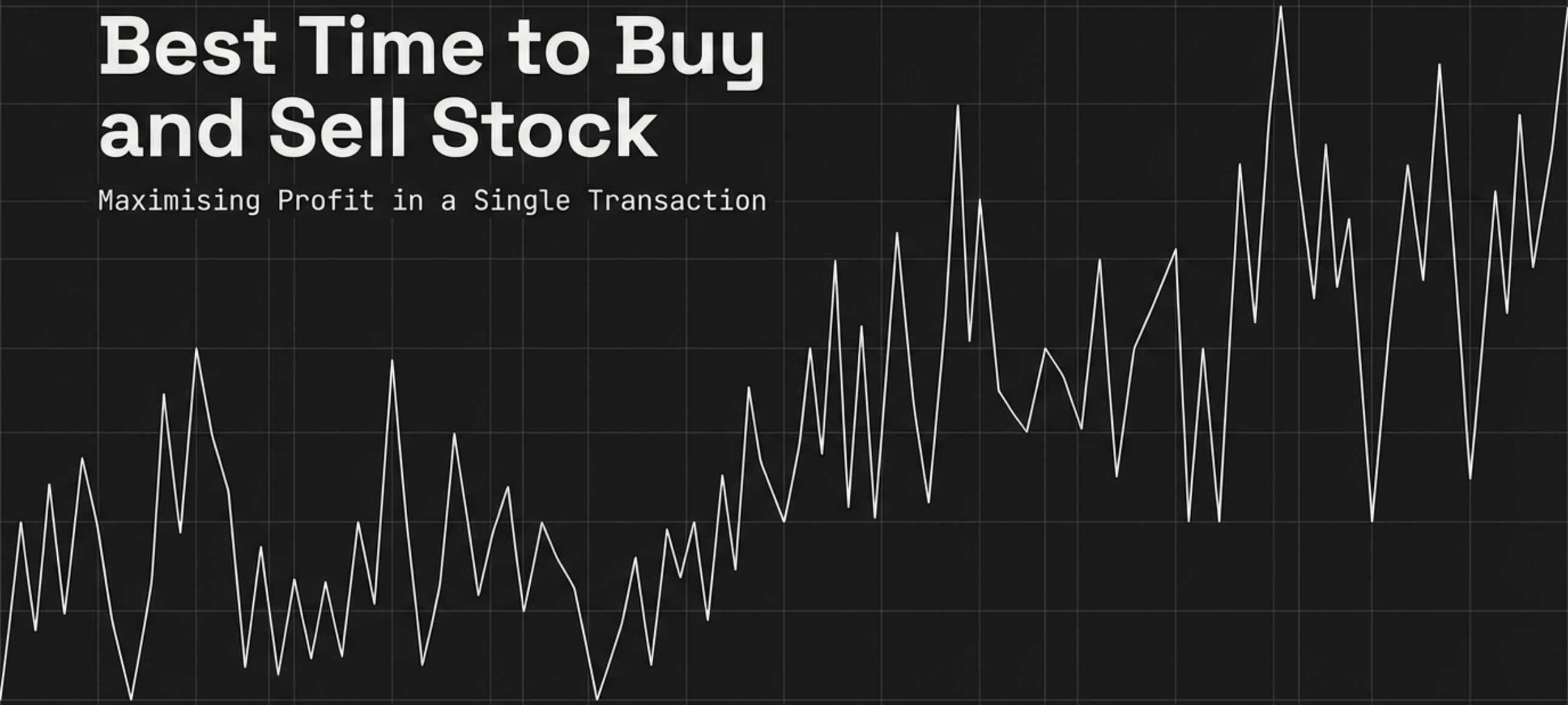
Best Time to Buy and Sell Stock

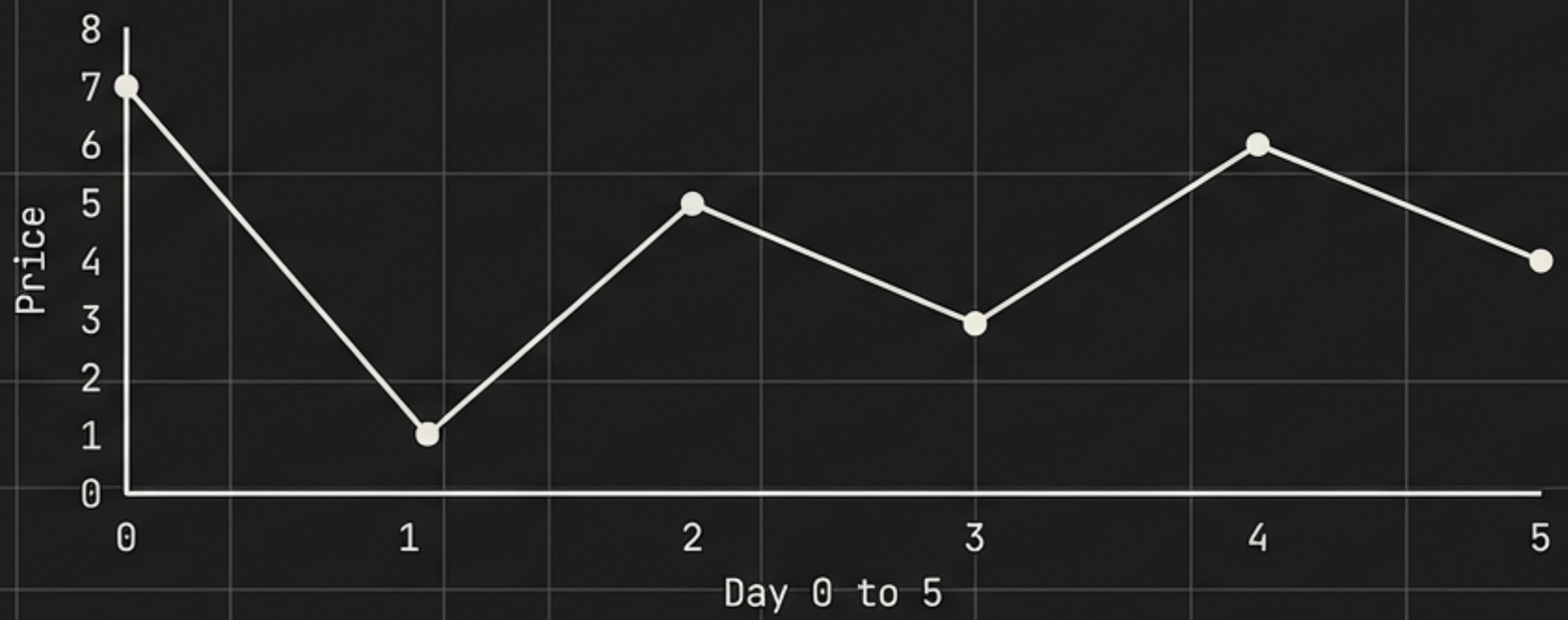
Maximising Profit in a Single Transaction



LEETCODE BLIND 75 | ARRAY SERIES

THE CHALLENGE

7	1	5	3	6	4



INPUT:

An array of daily stock prices.

GOAL:

One Buy. One Sell. Maximise Profit.

CONSTRAINT:

Cannot sell before buying.

TARGET:

Return max profit or 0.

THE INSTINCT: BRUTE FORCE

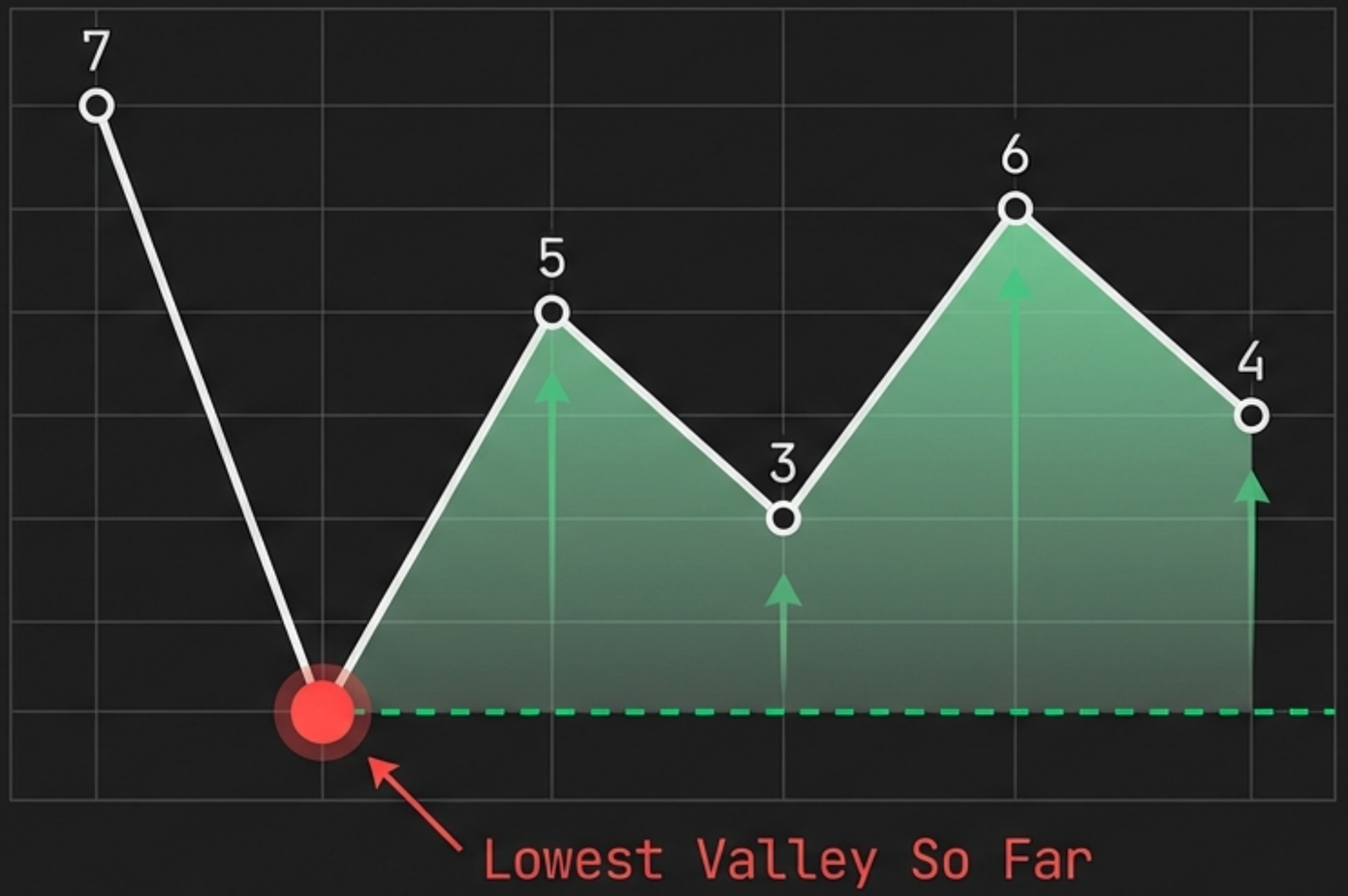
Check every pair.

Iterate: For every buy day i, check every sell day j.
Calculate: profit = prices[j] - prices[i].

The Bottleneck

- **Complexity:** $O(N^2)$
- **Issue:** Operations explode quadratically as the dataset grows. Too slow for real-world trading data.

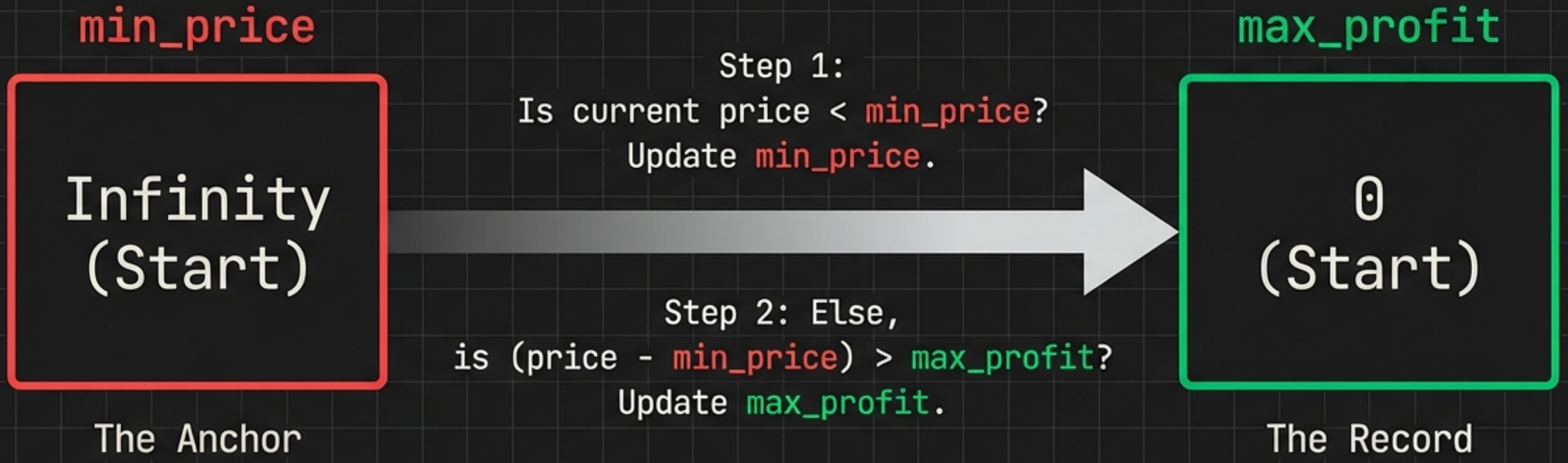
A SHIFT IN PERSPECTIVE



Single Pass Strategy

1. Don't look at pairs.
2. Traverse the timeline once.
3. Remember the lowest price seen so far.

The Logic: Kadane's Variation



We never look back. We only track these two values.

Walking the Timeline: Part 1

min_price: 1

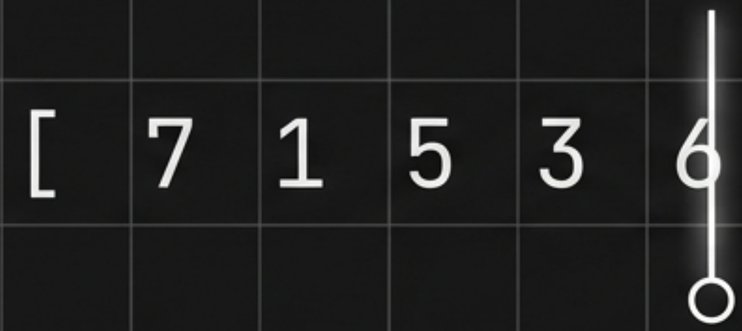
current_price: 5

calculation: 5 - 1 = 4

max_profit: 4

We found a valley at 1. Now at 5, the potential profit is 4. We store it.

Walking the Timeline: Part 2



min_price: 1 current_price: 6

calculation: 6 - 1 = 5

max_profit: 5

The dip to 3 didn't change min_price (3 > 1) and didn't beat max_profit (3-1 < 4). Ignored.

The Optimized Solution

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        min_price = float('inf')
        max_profit = 0

        for price in prices:
            if price < min_price:
                min_price = price
            elif price - min_price > max_profit:
                max_profit = price - min_price

        return max_profit
```

Complexity Analysis & Key Takeaways

Brute Force	Time: $O(N^2)$	Space: O(1)	Verdict: Too Slow
Optimised	Time: O(N)	Space: O(1)	Verdict: Optimal

- ✓ **Single Pass:** Iterate once.
- ✓ **Greedy approach:** Local optimality.
- ✓ **Memory efficient:** Only store min & max.

Foundation for LeetCode Array Series.