Best Time to Buy and Sell *Stock*

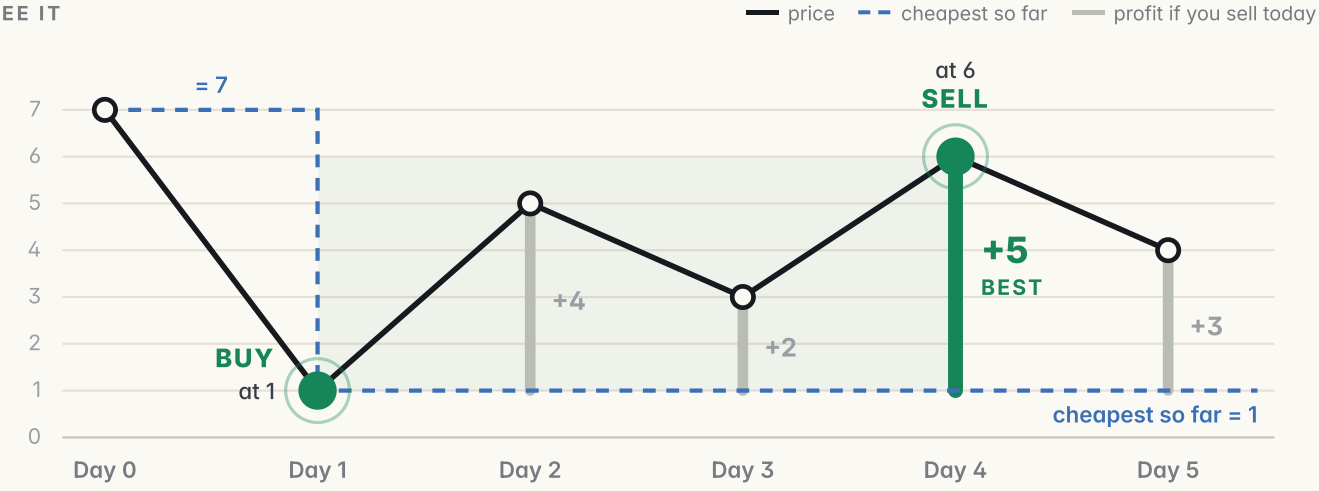
Buy once, sell later, make the most profit. Solved in one pass with a running minimum.

Easy | Arrays | Running minimum | One pass | O(n) time | O(1) space | LeetCode 121

The problem. `prices[i]` is a stock's price on day `i`. Pick one day to buy and a **later** day to sell. Return the largest profit you can make, or **0** if no trade makes money.

EXAMPLE
`[7, 1, 5, 3, 6, 4]` → **5**

SEE IT



The best day to buy is always *the cheapest day before you sell.*

So skip checking every pair. Walk left to right once, remember the lowest price so far, and on each day ask: "What if I sold today?"

THE ALGORITHM

1. **Start**
   `low = ∞, best = 0`
2. **Sell today?**
   `best = max(best, price − low)`
3. **Buy today?**
   `low = min(low, price)`

TIME	SPACE
O(n)	**O(1)**
brute force ~~O(n²)~~	two variables

THE SOLUTION

PYTHON

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        min_price = float("inf")  # cheapest so far
        max_profit = 0            # best profit so far

        for price in prices:
            # 1. sell today?
            max_profit = max(max_profit, price - min_price)
            # 2. buy today?
            min_price = min(min_price, price)

        return max_profit
```


Dry run, edge cases and follow-ups on page 2 →

01 DRY RUN · [7, 1, 5, 3, 6, 4]

DAY	PRICE	CHEAPEST BEFORE	SELL TODAY	BEST SO FAR
0	7	∞	–	0
1	1	7	−6	0
2	5	1	4	4
3	3	1	2	4
4	6	1	5	5
5	4	1	3	5

Green rows: a new best profit. The cheapest price stays at 1 from day 1 on. Answer: **5**.

02 WHY IT WORKS

Invariant. Before day `i`, `min_price` is the lowest price on days 0…i−1, and `max_profit` is the best trade that sells before day `i`.

As a 1-D DP, squeezed into two variables:

```
best = max(best, p[i] − min(p[0..i−1]))
```

Kadane view. Daily changes `[−6, 4, −2, 3, −2]`: the answer is the best run sum, `4 − 2 + 3 = 5`.

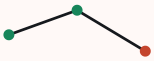
03 EDGE CASES TO TEST OUT LOUD

Only falls
`[7, 6, 4, 3, 1]`



No sale makes money → **0**. Never negative.

The trap
`[2, 4, 1]`



max − min = 4 − 1 = **3 ×**. The 1 comes after the 4. Correct: **2**.

Flat
`[3, 3, 3]`

Every profit is 0 → **0**.

One day
`[5]`



No later day to sell → **0**.

04 COMMON MISTAKES

- **Using max(prices) − min(prices).** It ignores order. You can't sell before you buy.
- **Starting min_price at 0.** Every "profit" becomes the full price. Start at ∞ or at `prices[0]`.
- **Letting the answer go negative.** Start `max_profit` at 0, which means "don't trade."
- **Nested loops.** O(n²) times out when there are up to 10^5 prices.

05 SAY THIS IN THE INTERVIEW

1. **Clarify.** One trade only? Sell strictly after buy? Return 0 if there is no profit?
2. **Brute force.** "Try every pair i < j: O(n²)."
3. **Insight.** "For each sell day, only the cheapest earlier price matters."
4. **Code it** in one pass, and state the invariant.
5. **Test** the example, a falling list and one day. "O(n) time, O(1) space."

06 FOLLOW-UPS: THE STOCK FAMILY

IF THEY CHANGE THE RULE TO…	IDEA	COST
Unlimited trades · 122	Add up every price rise from one day to the next.	O(n)
At most 2 trades · 123	Track 4 states: buy1, sell1, buy2, sell2.	O(n)
At most k trades · 188	The same states, repeated for each trade up to k.	O(n·k)
Cooldown after selling · 309	3 states: holding, just sold, resting.	O(n)
Fee on every sale · 714	Like unlimited trades, but subtract the fee at each sale.	O(n)
Return the days too	Save the index when the min changes; save the pair when the best changes.	O(n)

07 SAME SOLUTION, OTHER LANGUAGES

JAVA

```java
public int maxProfit(int[] prices) {
    int minPrice = prices[0], maxProfit = 0;
    for (int price : prices) {
        maxProfit = Math.max(maxProfit, price - minPrice);
        minPrice = Math.min(minPrice, price);
    }
    return maxProfit;
}
```

JAVASCRIPT

```javascript
function maxProfit(prices) {
    let minPrice = prices[0], maxProfit = 0;
    for (const price of prices) {
        maxProfit = Math.max(maxProfit, price - minPrice);
        minPrice = Math.min(minPrice, price);
    }
    return maxProfit;
}
```

C++: the same lines as Java, with `std::max` / `std::min`.