LeetCode 121. Best Time to Buy and Sell Stock

1. Problem Title & Link

- **121. Best Time to Buy and Sell Stock**
- https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

2. Problem Statement (Short Summary)

You are given an array prices where prices[i] is the price of a stock on the i^{th} day.
You want to **maximize your profit** by choosing a single day to **buy** one stock and a different day in the future to **sell** it.
Return the *maximum profit* you can achieve. If no profit is possible, return 0.

3. Examples (Input → Output)

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Input: prices = [7,6,4,3,1]
Output: 0
Explanation: No transactions possible (all decreasing).

4. Constraints

- 1 <= prices.length <= 10^5
- 0 <= prices[i] <= 10^4

5. Thought Process (Step by Step)

This is a **one-pass greedy optimization** problem
We want to **buy at the lowest price** seen so far and **sell at the best price** that comes after it.
Key Idea:

- Track the **minimum price so far (min_price)**.
- Track the **maximum profit so far (max_profit)**.
- For each price:
  - Profit if sold today = price - min_price
  - Update max_profit = max(max_profit, profit)
  - Update min_price = min(min_price, price)

At the end, max_profit gives the best possible result.

6. Pseudocode

```
min_price = prices[0]
max_profit = 0

for each price in prices:
   profit = price - min_price
   max_profit = max(max_profit, profit)
   min_price = min(min_price, price)

return max_profit
```

7. Code Implementation

✅ **Python**

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

        for price in prices:
            profit = price - min_price
            max_profit = max(max_profit, profit)
            min_price = min(min_price, price)

        return max_profit
```

✅ **Java**

```
class Solution {
    public int maxProfit(int[] prices) {
        int minPrice = prices[0];
        int maxProfit = 0;

        for (int price : prices) {
            int profit = price - minPrice;
            if (profit > maxProfit)
                maxProfit = profit;
            if (price < minPrice)
                minPrice = price;
        }
        return maxProfit;
    }
}
```

8. Time & Space Complexity

- **Time:** O(n) — one linear pass.
- **Space:** O(1) — constant extra variables.

9. Dry Run (Step-by-Step Execution)

👉 Input: prices = [7,1,5,3,6,4]

Day	Price	min_price	profit (price - min)	max_profit	Comment
1	7	7	0	0	initial buy
2	1	1	0	0	new min found
3	5	1	4	4	sell → profit=4
4	3	1	2	4	no better profit
5	6	1	5	5	new best profit
6	4	1	3	5	end

✅ Final Answer: **5**

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Greedy Single-Pass Optimization — keep track of min and max difference dynamically.	- Stock buy/sell problems - Maximum difference in arrays - Real-time min/max tracking	- Trying to do nested loops ($O(n^2)$) - Forgetting that "buy before sell" is mandatory - Resetting min_price incorrectly	🔷 Builds to **LeetCode 122 (Buy & Sell II)** and **LeetCode 123 (Two Transactions)** 🔷 Leads into **Dynamic Programming for Stock Problems (LC 188, 309, 714)** 🔷 Great base for **Kadane's-style logic** (similar to LC 53).

11. Common Mistakes / Edge Cases

- Initializing max_profit with -inf — incorrect when all losses exist (should return 0).
- Not updating min_price correctly when prices decrease continuously.
- Forgetting to handle single-element arrays.

12. Variations / Follow-Ups

- **LeetCode 122:** Buy and sell *multiple* times.
- **LeetCode 123:** At most *two* transactions.
- **LeetCode 309:** With *cooldown*.
- **LeetCode 714:** With *transaction fee*.