🎯 Question (LeetCode 122: Best Time to Buy and Sell Stock II)

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i` th day. You may complete **as many transactions as you like** (i.e., buy one and sell one share of the stock multiple times). **However, you must sell the stock before you buy again.** Return the maximum profit you can achieve.

Example:

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

🏷️ Best Time to Buy and Sell Stock II

📁 1. Definition and Purpose

- Maximize profit with unlimited buy-sell transactions.
- Each transaction must be completed (buy then sell).
- Useful in high-frequency trading and stock profit calculation.

📁 2. Syntax and Structure (Python)

```
# prices: list of integers representing stock prices per day
```

📁 3. Two Approaches

🧾 Approach 1: Brute Force (Recursive / Backtracking)

- Explore all possible sequences of buy-sell transactions.

```python
def max_profit_bruteforce(prices, start=0):
    max_profit = 0
    for i in range(start, len(prices)):
        for j in range(i+1, len(prices)):
            if prices[j] > prices[i]:
                profit = prices[j] - prices[i] + max_profit_bruteforce(prices, j+1)
                max_profit = max(max_profit, profit)
    return max_profit
```

- **Time Complexity:** O(2^n) (Exponential)
- **Space Complexity:** O(n) (Recursion stack)

Approach 2: Optimized (Greedy)

- Add profit for every ascending pair of days.
- O(n) time, O(1) space.

4. Optimized Pseudocode

```
profit = 0
for i in range(1, len(prices)):
    if prices[i] > prices[i-1]:
        profit += prices[i] - prices[i-1]
return profit
```

5. Python Implementation with Detailed Comments

```python
def max_profit(prices: list[int]) -> int:
    """
    Calculate maximum profit with unlimited transactions.
    """
    profit = 0  # Initialize total profit

    for i in range(1, len(prices)):
        if prices[i] > prices[i-1]:  # Profit possible if price increased
            profit += prices[i] - prices[i-1]  # Add difference to total profit

    return profit

# Example Usage
prices = [7,1,5,3,6,4]
print(max_profit(prices))  # Output: 7
```

6. Internal Working

- Traverse the array once.
- Whenever price[i] > price[i-1], sell for profit.
- Cumulatively sum all profitable transactions.

📂 7. Best Practices

- Avoid recursion for large input arrays.
- Use greedy approach for O(n) efficiency.
- Handle empty arrays by returning 0.

📂 8. Related Concepts

- Greedy algorithms
- Array traversal
- Stock trading strategies

📂 9. Complexity Analysis

- **Optimized Approach:**
  - Time: O(n)
  - Space: O(1)
- **Brute Force Approach:**
  - Time: O(2^n)
  - Space: O(n)

📂 10. Practice and Application

- LeetCode: 122 Best Time to Buy and Sell Stock II
- Useful in calculating maximum profit in multiple transactions, high-frequency trading, and financial simulations.